Министерство образования Российской Федерации

Нижегородский государственный университет
им.Н.И.Лобачевского

Высокопроизводительные параллельные вычисления на кластерных системах

Материалы третьего Международного
научно-практического семинара

13–15 ноября 2003 г.

Издательство Нижегородского госуниверситета
Нижний Новгород
2003

УДК 681.3.012:51
ББК 32.973.26–018.2:22
В 93

В93 **Высокопроизводительные параллельные вычисления на кластерных системах.** Материалы третьего Международного научно-практического семинара / Под ред. проф. **Р.Г. Стронгина.** Нижний Новгород: Изд-во Нижегородского госуниверситета, 2003. ??? с.

Сборник сформирован по итогам научного семинара, посвященного теоретической и практической проблематике параллельных вычислений, ориентированных на использование современных многопроцессорных архитектур кластерного типа. Семинар проходил в Нижнем Новгороде 13-15 сентября 2003 года.

Вошедшие в сборник материалы семинара представляют интерес для преподавателей и научных сотрудников высших учебных заведений, а также для инженеров, аспирантов и студентов вузов.

Отв. за выпуск к.ф.-м.н., доцент **В.А. Гришагин**

ББК 32.973.26–018.2:22

Поддержка семинара

РФФИ — Российский фонд фунтаментальных исследований

intel. — Компания Intel Technologies

NSTLab — Нижегородская лаборатория программых технологий

Министерство образования Российской Федерации (ФЦП «Интеграция» и НП «Университеты России»)

FASIE — Фонд содействия развитию малых форм предприятий в научно-технической сфере

13–15 сентября 2003 года Вычислительный Центр РАН, Институт математического моделирования РАН, Нижегородский государственный университет им. Н.И. Лобачевского провели в Нижнем Новгороде третий Международный научно-практический семинар и Всероссийскую молодежную школу «Высокопроизводительные параллельные вычисления на кластерных системах». Молодежная школа была организована совместно с Международной научной конференцией Parallel Computing Technologies (PaCT – 2003).

Главная направленность семинара и школы состояла в обсуждении основных аспектов организации высокопроизводительных вычислений в кластерных в кластерных компьютерных системах, активизации научно-практической деятельности исследователей в этой перспективной области развития современных средств вычислительной техники, обмене опытом учебно-образовательной деятельности при подготовке специалистов в области параллельных вычислений.

Проблематика семинара нацелена на рассмотрение следующих вопросов параллельных вычислений:

- принципы построения кластерных вычислительных систем;
- методы управления параллельных вычислениями в кластерных системах;
- параллельные алгоритмы решения сложных вычислительных задач;
- программные среды и средства для разработки параллельных программ;
- прикладные программные системы параллельных вычислений;
- методы анализа и оценки эффективности параллельных программ;
- проблемы подготовки специалистов в области параллельных вычислений.

В данный сборник включены материалы сообщений, которые представлены как в виде статей, так и в виде кратких тезисов. Материалы сборника упорядочены в алфавитном порядке по фамилии первого автора.

ОРГКОМИТЕТ СЕМИНАРА

Р.Г. Стронгин	председатель Оргкомитета, Первый проректор ННГУ, профессор, д.ф.-м.н.
Гергель В.П.	заместитель председателя Оргкомитета, профессор, д.т.н., ННГУ.
Батищев Д.И.	профессор, д.т.н., ННГУ.
Евтушенко Ю.Г.	директор Вычислительного центра РАН, чл.-корр. РАН.
Золотарев В.И.	директор Петродворцового телекоммуникационного центра СПбГУ
Крюков В.А.	зав. отделом Института Прикладной Математики РАН, профессор, д.ф.-м.н.
Нестеренко Л.В.	директор по стратегии и технологиям Нижегородской лаборатории Intel (INNL), к.ф.-м.н.
Сергеев Я.Д.	профессор, д.ф.-м.н., ННГУ, Калабрийский университет (Италия).
Четверушкин Б.Н.	директор Института математического моделирования РАН, чл.-корр. РАН.
Юнаковский А.Д.	зав. лаб. Института Прикладной Физики РАН, к.ф.-м.н.
Гришагин В.А.	ученый секретарь семинара, к.ф.-м.н., ННГУ

ПОРТАЛ ПОДДЕРЖКИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

К.Е. Афанасьев, А.В. Демидов

keafa@kemsu.ru , demid@kemsu.ru

За последнее десятилетие все более широкое применение в отечественных научных и производственных расчетах занимают вычисления на компьютерных кластерах. Конечно, фирменные SMP- и MPP-системы гораздо производительнее кластеров с таким же числом процессоров, однако их высокая стоимость делает фирменные системы недоступными большинству российских предприятий, институтов и университетов. С другой стороны, практически во всех организациях такого типа развернуты локальные компьютерные сети, объединяющие персональные компьютеры сотрудников, учебные классы и т.п. Многие локальные сети также подключены к глобальным сетям, в основном к Интернет. В то же время компьютерные ресурсы организаций практически не используются круглосуточно, более того их использование в учебном или производственном процессе редко превышает 12 часов в сутки. Это позволяет администраторам организовывать на базе существующих сетей компьютерные кластеры (преимущественно Linux-кластеры), которые могут функционировать в нерабочее время (например, ночью). Благодаря отсутствию необходимости приобретения дополнительного оборудования и свободному распространению программного обеспечения организация кластера не требует больших финансовых вложений.

Многие организации, а особенно университеты, имеют несколько компьютерных классов, на базе каждого из которых легко может быть организован однородный кластер. Причем может получиться так, что у организации возникнет излишек вычислительных ресурсов, которые она может отдавать и/или продавать сторонним пользователям. Зачастую бывает так, что сторонний пользователь не имеет возможности работать непосредственно на кластере, а обращается к его ресурсам посредством глобальных сетей. При этом возникает проблема: как организовать работу удаленных пользователей с минимальным риском для безопасности собственных ресурсов.

В большинстве случаев этот вопрос решается с помощью протокола SSH. Этот протокол хорош с точки зрения безопасности и простоты организации удаленного доступа, однако его недостатками являются интерфейс командной строки и зависимость набора команд от операционной системы удаленного ресурса. Если же мы организуем удаленный доступ через специализированный сервер, то мы обеспечим и единый интерфейс при доступе к различным ресурсам, и можем предоставить пользователю графический интерфейс посредством клиентского приложения.

В процессе проектирования системы удаленного доступа появлялись все новые и новые задачи, решение каждой из которых повысило бы комфорт пользователя при работе с высокопроизводительными ресурсами. Дивергенция этих задач и их группировка в конце концов позволили выделить пять основных направлений деятельности, а понятием, объединяющим эти задачи, стал Вычислительный Портал.

Портал в данном контексте позиционируется как замкнутый аппаратно-программный комплекс, содержащий в себе полный набор необходимых инструментов для обеспечения всех этапов расчетов на вычислительных системах с параллельной архитектурой от методических указаний по распараллеливанию алгоритмов до высокопроизводительных ресурсов и средств удаленного доступа к ним. В структуру портала входят следующие основные компоненты:

1. Вычислительные ресурсы.

   Это аппаратная часть портала, которая может быть представлена как SMP- и MPP-системами заводской сборки, так и компьютерными кластерами различной вычислительной мощности. На самых мощных системах можно запускать научные расчеты, а на более слабых, например, исследовать распараллеливание численных методов. Необходимым требованием к вычислительному ресурсу является поддержка им возможности удаленного управления посредством rshell (remote shell).
2. Средства удаленного доступа.

Это основной программный компонент портала, который обеспечивает удаленный доступ пользователей к вычислительным ресурсам.

3. Библиотеки распараллеленных процедур и функций.

   Пользователь портала должен думать о результатах своих расчетов, а не о том, как лучше ему распараллелить тот или иной блок вычислений. Чем больше «строительных блоков» для создания параллельных программ ему будет предоставлено, тем проще и быстрее будет сделать расчет и получить результат. В этом компоненте могут быть представлены как широко известные математические библиотеки (например, SCALAPACK), так и библиотеки процедур, характерных для какой либо конкретной области вычислений (например, процедуры, типичные для решения задач гидродинамики).

4. Средства отладки параллельных программ.

   Отлаживать распараллеленные алгоритмы и программы непосредственно на высокопроизводительном вычислительном ресурсе как правило не рационально в силу высокой стоимости или дефицита процессорного времени ресурса. Поэтому необходимо предоставить возможность пользователю отлаживать программы на своем рабочем месте, а на кластер отправлять уже не содержащие очевидных ошибок программы. Работы по созданию средств отладки ведутся во многих ВУЗах России и есть несколько подходов к решению этой задачи. Мы пошли по пути эмуляции многопроцессорности средствами псевдо-многозадачной ОС семейства Windows.

5. Информация и документация.

   Чтобы портал в полной мере отрабатывал свое название и был «информационно-вычислительным», он должен содержать информацию. В Интернете выложено довольно много документации по вопросам распараллеливания алгоритмов и работы в средах параллельных вычислений. Предполагается создать сайт, на котором размещать ссылки на отечественные информационные ресурсы, перевод зарубежных статей и руководств, а также собственные информационные материалы. Кроме того, непременно нужно дать пользователю подробные

инструкции по работе с каждым элементом портала, а особенно со средствами отладки и удаленного доступа.

ПО удаленного доступа

Согласно опросу, проведенному среди потенциальных пользователей системы, пользователь хочет, чтобы система позволяла:

1. Работать с сервером по протоколу TCP/IP;
2. Создавать новые проекты (расчетные программы) и копировать исходные коды программ с машины клиента на сервер;
3. Компилировать и перекомпилировать выбранные проекты на выбранном кластере, сообщать результаты компиляции, возвращать ошибки компиляции;
4. Запускать экземпляры расчетных программ на выбранном кластере;
5. Уничтожать выбранные экземпляры расчетных программ;
6. Сообщать об успешном или неуспешном завершении расчета;
7. Копировать файлы с результатами расчета с сервера на машину клиента;
8. Возвращать ошибки времени исполнения;
9. Сохранять настройки интерфейса и параметров запуска расчетов;
10. Удалять выбранные проекты и файлы.

Эти требования полностью были заложены в функциональные возможности разрабатываемой системы. Дополнительно была предусмотрена функция вызова хранимых на сервере расчетных программ и поддержка серий вычислительных экспериментов для последующей интеграции данной системы с пакетом прикладных программ «Акорд», созданном в ЦНИТ КемГУ для численного решения задач гидродинамики [1].

Исходя из приведенных требований и анализа предметной области, можно сделать вывод о том, что наиболее рациональным способом реализации поставленной задачи является распределенное приложение архитектуры «клиент–сервер».

В структуре «Вычислительного портала», сервер удаленного доступа является центральным компонентом. В его функции входит, с одной стороны, работа с клиентами, а с другой стороны – управление доступными вычислительными ресурсами в соответствии с запросами клиентов.

Сервер – это набор процессов, запущенных на выделенном ПК. Один из этих процессов работает постоянно. Это базовый процесс, прослушивающий сеть и принимающий входящие запросы на соединение от клиентов. Остальные процессы работают не постоянно, а создаются базовым или другим процессом для выполнения некоторого задания, либо системного, либо возникшего в результате запроса клиента.

В функции клиента входит работа с сервером для обеспечения доступа пользователя к ресурсам «Вычислительного портала». В структуре «Вычислительного портала» клиент является инвариантной частью и может быть представлен различными способами. В текущей реализации клиент выполняется в виде приложения под ОС Windows 98/2000/XP. Язык разработки – Delphi, среда разработки – Borland Delphi 6. В дальнейшем перспективным видится переход к клиенту, основанному на веб-браузере, что потребует введения дополнительных компонентов в структуру сервера без изменения общей логики его функционирования.

Для взаимодействия клиента и сервера между собой разрабатывается собственный протокол прикладного уровня (далее – протокол), работающий поверх протокола TCP/IP. Изначально планировалось использовать текстовый протокол, работающий по принципу протокола передачи почты POP3. Однако оказалось, что текстовые протоколы не имеют преимуществ перед бинарными, а при использовании последних к тому же отсутствует необходимость парсинга (синтаксического разбора) команд.

Средства отладки: эмулятор многопроцессорности

При разработке любого приложения значительные усилия тратятся на отладку исходного кода. Естественно, что этот процесс требует значительных временных затрат и затрат вычислительных ресурсов. Процессорное время высокопроизводительных ресурсов, как правило, дефицитное и, следовательно, дорогое. Отлаживать приложения на высокопроизводительном кластере нерационально, гораздо эффективнее запускать на счет заведомо рабочую программу.

Чтобы отладить все ошибки (как алгоритмические, так и синтаксические), требуется кластер, по архитектуре сходный с тем, на котором предполагается запускать задачу. Поскольку не у каждого есть возможность собрать собственный кластер, то в этом случае

можно обойтись персональным компьютером, эмулируя на нем многопроцессорную среду.

Главным требованием, которое ставилось перед созданием эмулятора и которое неукоснительно выполнялось, это переносимость кода из среды эмуляции в среду выполнения без каких-либо изменений. То есть эмулятор понимает синтаксис вызова всех процедур языка Fortran и библиотеки MPICH также, как они будут пониматься высокопроизводительным кластером. Текущая версия эмулятора [2] рассчитана на работу с программами, написанными на языке Fortran77 с использованием интерфейса обмена сообщениями MPICH версии 1.2.2. Т.к. большинство рабочих станций в настоящее время работают под управлением ОС семейства Windows, то текущая реализация эмулятора рассчитана на работу под управлением именно этих ОС.

Последние ОС семейства Windows являются псевдо-много-задачными и позволяют на одном процессоре запускать несколько ветвей параллельного приложения, которые будут работать в режиме конкуренции за процессорное время. Это позволяет запустить несколько процессов в системе, каждый из которых будет являться одной из ветвей параллельного приложения.

Передача сообщений между процессами идет через файлы на жестком диске. Недостатком такого метода является низкая скорость обмена данными процессора с жестким диском. Тем не менее, не следует забывать, отладка программ, как правило, выполняется с небольшим набором данных и скорость на данном этапе не является критическим параметром. С другой стороны возможность просмотра этих файлов дает дополнительный инструмент для поиска ошибок в алгоритме и программе. Данный подход к эмулированию многопроцессорности был предложен в дипломной работе Д.А. Сучковой, выполненной в Уфимском Государственном авиационном техническом университете (УГАТУ) под руководством профессора В.П. Житникова.

Основой эмулятора является модуль с описанием процедур библиотеки MPICH, который подключается к эмулируемой программе и обеспечивает необходимые действия при вызове из программы функций MPICH, для этого приходится переопределять функции библиотеки MPICH. В модуль включены функции инициализации

(MPI_Init, MPI_Comm_Rank, MPI_Comm_Size, MPI_Comm_Finalize), которые встречаются в любой параллельной программе, простые функции пересылки данных (MPI_Send и MPI_Recv), и функции групповых коммуникаций (MPI_Reduce, MPI_Scatter и другие).

Эмулятор реализован в виде Win32 приложения, которое после ввода пользователем параметров самостоятельно компилирует и запускает соответствующее число ветвей параллельного приложения.

Информационные материалы

Этот раздел портала предполагает создание сайта поддержки параллельных вычислений. Здесь планируется размещать информацию по всем проблемам данного направления: от теории численных методов и их распараллеливания до практики отладки распределенных программ. В Интернете уже сейчас есть много информации по данной тематике.

В нашем портале будут размещаться копии русских документов, переводы иностранных и собственные материалы. Одним из первых будет электронный учебно-методический комплекс «Многопроцессорный вычислительные системы и параллельное программирование» («МВС и ПП»), созданный на кафедре ЮНЕСКО по НИТ К.Е. Афанасьевым, С.В. Стуколовым, А.В. Демидовым и В.В. Малышенко[3]. Данный комплекс официально зарегистрирован в Реестре баз данных Российского агентства по патентам и товарным знакам (Роспатент, свидетельство № 2003620094).

Те компоненты портала, которые существуют к настоящему моменту, пока что реализованы только на уровне прототипов и не могут быть открыты для публичного использования. Для начала активного использования компонентов портала, необходимо:

- завершить реализацию системы удаленного доступа с базовым набором функций;
- усовершенствовать статистику, выдаваемую эмулятором о ветвях параллельного приложения, перевести ее в графический вид;
- увеличить базу данных электронной документации по параллельным вычислениям;
- разработать и реализовать информационный сайт портала.

По завершению первой очереди работ, вычислительный портал будет готов к использованию в процессе подготовки специалистов по параллельным вычислениям на МФ КемГУ

После того, как будут реализованы поддержка защищенных протоколов передачи данных по глобальным сетям; поддержка вызова клиентом процедур и функций, хранимых на сервере; и поддержка серий экспериментов (запуск одной расчетной программы с различными исходными данными в автоматическом режиме) – система будет открыта для доступа пользователям, сторонним по отношению к КемГУ.

Построенные модели вполне допускают подобное расширение функциональности, и этот процесс пройдет в основном путем добавления нового программного кода с минимальным исправлением уже написанного.

Литература

1. Афанасьев К.Е., Гудов А.М. Информационные технологии в численных расчетах. Кемерово, 2001. 203 с.
2. Демидов А.В., Сидельников К.В. Эмуляция параллельной обработки данных на персональном компьютере // XLI Международная научная студенческая конференция «Студент и научно-технический прогресс». Сб. трудов. Новосибирск, 2003. С. 110-111.
3. Многопроцессорные вычислительные системы и параллельное программирование / Афанасьев К.Е., Стуколов С.В., Демидов А.В., Малышенко В.В. Кемерово, 2003. 181 с.

ИСПОЛЬЗОВАНИЕ OPENMP ПРИ РАСЧЕТЕ ВЛИЯНИЯ ДЕНЕЖНОЙ ПОЛИТИКИ ЦЕНТРАЛЬНОГО БАНКА НА ПРОИЗВОДСТВО

М.Н. Банникова, Д.А. Сивков

Удмуртский госуниверситет, г.Ижевск

Описание модели

Одним из основных факторов оказывающих воздействие на экономику является денежная политика Центрального банка по отношению к коммерческим банкам. Влияние денежной политики Центрального банка на производство можно представить в виде следующей схемы (см. [1])

Для поддержания финансовой деятельности коммерческих банков (КБ) Центральный банк наделяет их кредитом под процент r^B, который определяет все дальнейшие финансовые операции КБ. Прибыль КБ зависит от прибыли производителей и торговых посредников. КБ стремятся максимизировать свою прибыль, поэтому возникает две оптимизационные задачи (в обозначениях [1]): задача на максимум прибыли от кредитов (r_p) производителям

$$\frac{(r_p - r^B)}{\lambda} \int_0^{\frac{\lambda p_0}{\lambda + r_p}} y\eta(y)dy + \\ + \frac{r^B(1-\xi-\xi_2)}{\lambda} \int_{\frac{\lambda p_0}{\lambda + r_p}}^{p_0} \left\{ \ln\left(\frac{p_0 - y + wy}{wy} \right) + \frac{y - p_0}{p_0 - y + wy} \right\} y\eta(y)dy \to \max_{0 \le r_p < r^*}, \tag{1}$$

где n – количество видов производственных факторов,

$$\eta(y) = \int_0^{\frac{y}{p_1}} \int_0^{\frac{y - p_1 x_1}{p_2}} \dots \int_0^{\frac{y - p_1 x_1 - \dots - p_n x_n}{p_n}} \zeta(x_1) \dots \zeta(x_{n-1}) \zeta(x_n) dx_1 \dots dx_{n-1} dx_n \; -$$

распределение мощностей по себестоимости продукта,
$\zeta(x)$ – плотность распределения мощностей по технологиям,
x – вектор затрат на выпуск единицы продукции
и задача на максимум прибыли от кредитов (r_T) торговым посредникам

$$\frac{(\widetilde{p}_0 - p_0)Y(w, r_p^*, p_0)}{((\widetilde{p}_0 - p_0)Y(w, r_p^*, p_0) + \gamma_2 C_T)}\left((r_T - r^B)\frac{\widetilde{u}\varphi(r_T)}{(\lambda_T - r_T)_+}\times\right.$$
$$\times\min\left\{1, \left(\frac{\lambda_T r_T}{\Delta_T(\lambda_T + \Delta_T - r_T)_+}\right)^{\frac{-\lambda_T}{\lambda_T+\Delta_T}}\right\}+$$
$$+ r^B(1-\xi-\xi_3)\left(\frac{1}{\lambda_T+\Delta_T}\left[\ln\left(\frac{\lambda_T r_T}{\Delta_T(\lambda_T+\Delta_T-r_T)_+}\right)\right]_+ + \right. \quad (2)$$
$$\left.\left.+\frac{1}{\lambda_T}\left(\left(\frac{\lambda_T r_T}{\Delta_T(\lambda_T+\Delta_T-r_T)_+}\right)^{\frac{-\lambda_T}{\lambda_T+\Delta_T}} - 1\right)_-\right)\right) \to \max_{0\le r_T\le\lambda^*},$$

при ограничениях

$$(\widetilde{p}_0 - p_0)Y(w, r_p^*, p_0)\left(\frac{r_T\widetilde{u}\varphi(r_T)}{((\widetilde{p}_0 - p_0)Y(w, r_p^*, p_0) + \gamma_2 C_T)(\lambda_T - r_T)_+}\times\right.$$
$$\times\min\left\{1, \left(\frac{\lambda_T r_T}{\Delta_T(\lambda_T+\Delta_T-r_T)_+}\right)^{\frac{-\lambda_T}{\lambda_T+\Delta_T}}\right\}-$$
$$-\zeta_T\left(\left(\gamma_0 + \frac{\gamma_1 C_T\varphi(r_T)}{((\widetilde{p}_0 - p_0)Y(w, r_p^*, p_0) + \gamma_2 C_T)(1+\varphi(r_T))}\right)-\right. \quad (3)$$
$$\left.\left.-\left(-\frac{\widetilde{u}\varphi(r_T)}{(\widetilde{p}_0 - p_0)Y(w, r_p^*, p_0) + \gamma_2 C_T}\right)\right) \le 0,$$

где

$$\varphi(r_T) = \sqrt{\frac{\gamma_1 C_T}{\widetilde{u}\frac{\Delta_T}{\lambda_T}\left(1+\left(\ln\left(\frac{\lambda_T r_T}{\Delta_T(\lambda_T+\Delta_T-r_t)_+}\right)_+\right)\right)}},$$

$$Y(w, r_p^*, p_0) = \int_0^{p_0}\left(1-\left(\frac{wy}{wy+p_0-y}\right)\theta\left(r_p^* - \frac{p_0-y}{y}\right)\right)\eta(y)dy$$

Более полное описание всех используемых параметров и функций может быть найдено в [2].

Алгоритмизация поставленных задач

Для поиска решения задачи (1) применяется метод золотого сечения. Вычисление определённых интегралов осуществляется по правилу Рунге для составной квадратурной формулы Симпсона. Для вычисления многомерных интегралов составляется рекурсивная процедура.

Поиск решения задачи (2)–(3) осуществляется с помощью метода штрафных функций и метода простого поиска.

В процессе решения поставленных задач требуется многократное вычисление значений максимизируемых функций. На эти вычисления приходятся значительные временные затраты. Поэтому возникла необходимость в распараллеливании последовательной программы.

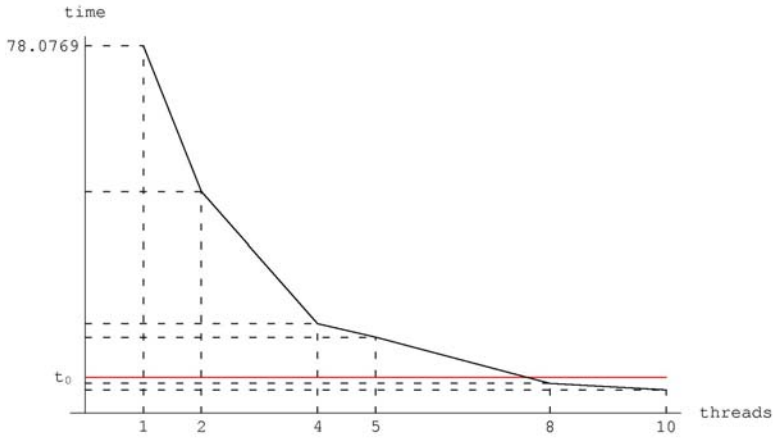
Как отмечалось выше, для вычисления многомерных интегралов составляется рекурсивная процедура. Но в связи с возникшими проблемами при распараллеливании было решено найти другой подход к вычислению многомерных интегралов. Выбор остановился на статистических методах, которые удобны для распараллеливания.

Средства вычислительной техники

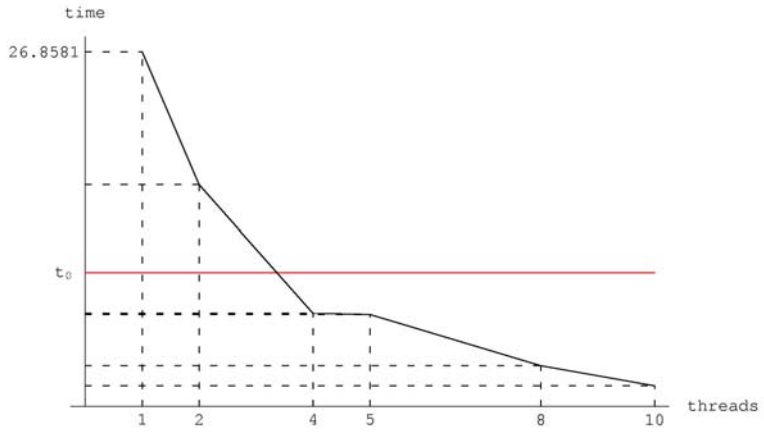
Параллельная программа была написана на языке C-OpenMP и транслировалась компилятором Intel C compiler 7.1.

Для сравнения времени работы последовательной и параллельной программ были проведены тестовые расчеты на кластере PARC Удмуртского госуниверситета с использованием предоставленного для тестирования компанией Intel компьютера SPSH4 (4 процессора Intel XeonMP 2GHz 2Mb).

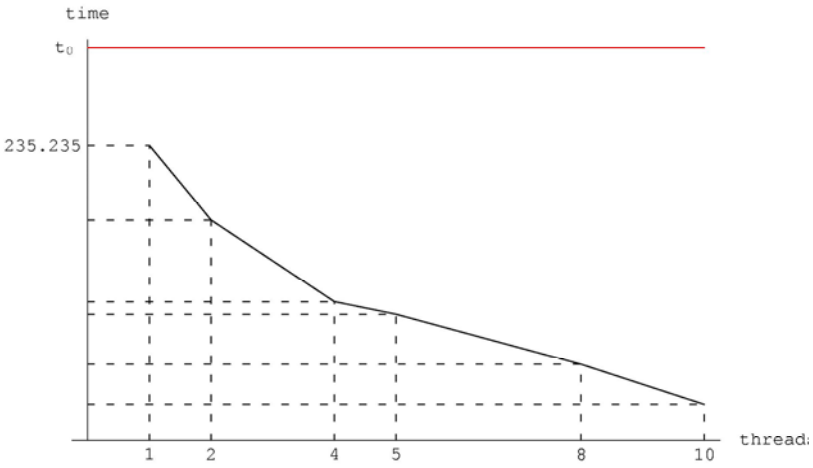
Сравнительный анализ



В случае, когда $n = 1$, т.е. подынтегральная функция является одномерным интегралом, параллельная программа будет эффективна, если число потоков больше 8. Красной линией показано время работы последовательной программы с рекурсивным вычислением интегралов.



Похожая ситуация возникает, когда $n = 2$. В этом случае число потоков должно быть не меньше 4.



Иная картина возникает в случае $n = 3$. Здесь, даже когда параллельная программа выполняется последовательно (число потоков равно 1), время её выполнения меньше, чем время выполнения последовательной программы. Это может объясняться иным методом вычисления интегралов. Однако и в этом случае для одного потока временные затраты всё равно остаются существенными. Поэтому, увеличивая число потоков можно добиться значительного уменьшения времени выполнения параллельной программы.

Литература

1. Автухович Э.В., Петров А.А., Поспелов И.Г., Шананин А.А. Исследование с помощью математической модели влияния на производство денежной политики центрального банка http://isir.ras.ru/ win/ db/show.
2. Автухович Э.В., Шананин А.А. Модель отрасли производства в условиях дефицита оборотных средств // Математическое моделирование , 2000. Т.12, №7. С.102-126.
3. Introduction to OpenMP http://www.llnl.gov/.

АССОЦИАТИВНАЯ ВЕРСИЯ АЛГОРИТМА ПОИСКА В ГЛУБИНУ*

Т.В. Борец

ИВМиМГ, г. Новосибирск
borets@ssd.sscc.ru

Введение

Поиск в глубину – широко распространенный метод систематического обхода графа. Он используется для решения многих задач исследования операций, искусственного интеллекта и других разделов информатики. Последовательный алгоритм хорошо известен и описан в книгах по теории графов и информатики [1–3]. Этот алгоритм выполняется за время $O(n+m)$, где n – число вершин графа, а m – число его ребер. Параллельная реализация этого алгоритма на MIMD – архитектуре приведена в [4, 5].

В данной работе представляется алгоритм, выполняющий поиск в глубину на абстрактной модели ассоциативного параллельного процессора с вертикальной обработкой данных (STAR-машина). Эта модель относится к классу SIMD-архитектур и была представлена в [6].

В начале будет описана модель, на которой реализован алгоритм, затем будет предложен сам алгоритм и обоснование его корректности.

Модель STAR-машины

STAR-машина состоит из

- последовательного устройства управления, в котором записаны программа и скалярные константы;
- матричной памяти;
- устройства ассоциативной обработки, состоящего из p одноразрядных процессорных элементов ($p = 2^i$).

Для этой архитектуры в работе [6] был разработан язык высокого уровня STAR, поддерживающий язык PASCAL. Чтобы моделировать обработку информации в матричной памяти в языке STAR имеются переменные типов данных **slice**, **word** и **table**. Переменная типа **slice** (далее слайс) моделирует доступ к таблице по столбцам, а переменная

* Работа выполнена при поддержке Российского фонда фундаментальных исследований (грант № 03-01-00399)

типа **word** – доступ по строкам. С каждой переменной типа **table** ассоциируется бинарная таблица из n строк и k столбцов, где $n \le p$. Также в язык были добавлены операции над этими типами данных. Мы приведем только те операции, которые необходимы для изложения алгоритма.

Пусть X и Y – слайсы, i – целочисленная переменная. Тогда *SET(Y)* записывает в слайс Y все единицы; *CLR(Y)* записывает в слайс Y все нули; *Y(i)* выделяет i-ю компоненту в слайсе Y ($1 \le i \le p$); *FND(Y)* выдает порядковый номер i позиции первой (или старшей) единицы в слайсе Y, где $i \ge 0$; *STEP(Y)* выдает такой же результат, как и операция *FND(Y)* и затем обнуляет старшую единицу. Общепринятым способом вводятся предикаты *ZERO(Y)* и *SOME(Y)*, а также следующие побитовые логические операции: *X and Y*, *X or Y*, *not X*, *X xor Y*.

Для переменных типа **word** выполняются те же операции, что и для переменных типа **slice**. Для переменных типа **table** определены следующие операции: *COL(i,T)* и *ROW(i,T)*, которые выделяют из таблицы T i-й столбец и i-ю строку, соответственно.

Следуя Фостеру [7], мы допускаем, что каждая элементарная операция STAR-машины выполняется за единицу времени. Поэтому сложность алгоритма определяется числом элементарных операций, которые выполняются в наихудшем случае.

Для реализации алгоритма нам потребуется следующая базовая процедура из [8], которая выполняется на STAR-машине за время пропорциональное числу битовых столбцов таблицы T.

Процедура MATCH(T: table; X: slice; v: word; var Z: slice) определяет позиции тех строк таблицы T, которые совпадают со словом w. Процедура возвращает слайс Z, в котором *Z(i)='1'* тогда и только тогда, когда *ROW(i, T)=w* и *X(i)='1'*.

Необходимые определения

Напомним некоторые определения из теории графов.

Ориентированным графом (орграфом) G называется пара множеств (V, E), где $V=\{1,\ldots, n\}$ – конечное множество вершин, E – множество дуг ($E \subseteq V \times V$).

Пусть $e=(u, v) \in E$ – дуга графа, тогда вершину v будем называть головой дуги e, а вершину u – **хвостом**.

Подграфом для графа G называется такой граф $H=(W, U)$, у которого $W \subseteq V$ и для любых $x,y \in W$, если $(x, y) \in E$, то $(x, y) \in U$.

Поиск в глубину – метод систематического прохождения (посещения) вершин графа. Он использует следующую стратегию: идти «вглубь», пока это возможно (есть исходящие дуги в непройденные вершины), и искать другой путь, когда таких дуг нет. Так делается, пока не обнаружены все вершины, достижимые из исходной. В порядке обхода каждой вершине приписывается *M*-номер и строится **дерево поиска в глубину**.

В матричной памяти STAR-машины орграф задается парой таблиц *left* и *right*, так что каждой дуге $(u, v) \in E$ соответствует пара $<u, v>$. Также используется таблица *code*, в *i*-ой строке которой записан двоичный код вершины *i*. Подграф и дерево поиска в глубину задаются слайсами, в которых соответствующие дуги помечаются ‘1’.

Ассоциативная версия алгоритма поиска в глубину

Структура данных

Алгоритм представлен в виде процедуры, записанной на языке STAR. Процедура использует следующие входные параметры:

- таблицы *left* и *right* задают множество дуг орграфа;
- таблица *code* хранит двоичные коды вершин орграфа;
- слово *root* хранит двоичный код вершины, с которой начинается поиск.

Алгоритм строит дерево поиска в глубину, нумерует вершины в порядке их обхода и возвращает подграф, состоящий из вершин, недостижимых из *root*:

- таблица *NV* хранит *M*-нумерацию вершин графа;
- слайс *T* задает дерево поиска в глубину (в T ‘1’ отмечены позиции тех и только тех дуг, которые принадлежат дереву);
- слайс *X*, в котором помечены позиции дуг, ведущих в непронумерованные вершины (после завершения работы процедуры задает подграф, содержащий только те вершины, которые недостижимы из *root*).

Также нам понадобятся следующие вспомогательные переменные:

- таблица *LIFO*, моделирующая стек (после нумерации очередной вершины в таблицу добавляется слайс, в котором ‘1’ отмечены позиции дуг, ведущих из этой вершины в непронумерованные);

- целочисленная переменная *lif*, хранящая глубину стека;
- целочисленная переменная *ord*, хранящая текущий *M*-номер;
- слово *node*, хранящее двоичный код вершины, из которой продолжается обход.

Идея использовать стек была представлена в [9].

Идея алгоритма

1. Инициализация переменных;
2. Нумерация текущей вершины и вычисление позиций тех дуг, которые ведут из нее в непронумерованные вершины;
3. Далее возможны случаи:
   3.1. если таких дуг нет и текущая вершина не является корневой, тогда поднимаемся на одну вершину выше по стеку и возвращаемся к шагу 3;
   3.2. если из текущей вершины есть такие дуги, тогда выбираем дугу, отмечаем ее в дереве и голову этой дуги делаем текущей вершиной, остальные такие дуги помещаем в стек и возвращаемся к шагу 2;
   3.3. если таких дуг нет и вершина корневая, тогда заканчиваем обход.

Теперь приведем процедуру, реализующую этот алгоритм.

```
Procedure DFS(left, right: table; code: table; root:
word; Var NV: table; Var T, X: slice);
Var    LIFO: table; {моделирует стек}
       lif,         {хранит глубину стека}
       ord,         {хранит текущий M-номер}
       i:     integer;
       Y, Z: slice{left};
       U, V: slice{code};
       w, node: word;
1 Begin
2      SET(X);      SET(Z) CLR(T);      SET(U);
3      ord:=1;      lif:=1;             node:=root;
4      repeat
  {Нумерация текущей вершины.}
5             MATCH(code, U, node, V);         i:=FND(V);
6             w:=ROW(ord, code);
```

```
7             ROW(i, NV):=w;      ord:=ord+1;
 {Поиск дуг, ведущих из вершины node в непронумерованные вершины.}
8             MATCH(right, Z, node, Y);
9             X:=X and (not Y);
10            MATCH(left, X, node, Y);
 {Если таких дуг нет, и текущаяя вершина не равна root, то поднимаемся по
 стеку.}
11            while ZERO(Y) and (lif>1) do
12            begin
13                  lif:=lif-1;
14                  Y:=COL(lif, LIFO);
15                  Y:=Y and X;
16            end;
 {Если есть дуга из пронумерованной вершины в непронумерованную, то
 заносим ее в дерево и голова этой дуги становится текущей вершиной.}
17            if SOME(Y) then
18            begin
19                  i:=STEP(Y);
20                  T(i):= '1';
21                  COL(lif, LIFO):=Y;  lif:=lif+1;
22                  node:=ROW(i, right);
23            end;
24      until lif>1;
25 End;
```

Теорема. *Пусть ориентированный граф G = (V, E) задан ассоциацией матриц left и right. Пусть code – матрица, в i-й строке которой записан двоичный код вершины с номером i и root – двоичный код вершины, с которой начинается поиск в глубину. Тогда процедура DFS(left, right, code, root, NV, T, X) возвращает матрицу NV, в i-й строке которой записан двоичный код M-номера вершины i, слайс T, в котором '1' помечены дуги, принадлежащие дереву поиска в глубину, и слайс X, в котором '1' отмечены дуги подграфа, состоящего из всех вершин, недостижимых из root. Время работы процедуры $O(n_r \cdot \log n)$, где n_r – число вершин, достижимых из вершины root.*

Доказательство.

Будем проводить индукцией по n_r – числу вершин, достижимых из *root*.

*Базис***.**

Пусть n_r=1. Рассмотрим подробнее работу процедуры. После выполнения строк (5-10) вершина *root* получила *M*-номер, который был записан в соответствующую строку таблицы *NV*, из слайса *X* были удалены позиции всех дуг, заходящих в вершину *root* и в слайсе *Y* ‘1’ отмечена позиция единственной дуги, выходящей из вершины root.

Поэтому условие цикла (11–16) нарушается, но выполняется условие в строке 17. После выполнения строк (19–22) позиция этой дуги заносится в слайс *T*. В таблицу *LIFO* добавляется пустой столбец. Текущей вершиной *node* становится голова добавленной дуги. Так как *lif = 2*, то процедура продолжает свою работу со строки 5.

Происходит нумерация вершины *node,* из слайса *X* удаляются все дуги, заходящие в вершину *node*, слайс *Y* пуст. Так как *lif = 2*, то выполняется тело цикла (13–15): *lif* становится равным 1, а слайс *Y* остается пустым. После проверки условий в строках 11, 17 и 24 процедура останавливается. Слайс *T* содержит только одну ‘1’, которая соответствует дуге *(root, node)*, из слайса *X* удалены позиции всех дуг, заходящих в вершины *root* и *node*.

Шаг индукции.

Пусть для $n_r = k$–1 теорема уже доказана, докажем для *k*. После выполнения строк 5–10, как уже показывалось, в соответствующую строку таблицы *NV* записан M – номер вершины *root,* позиции всех дуг, заходящих в эту вершину, удалены из слайса *X* и позиции всех дуг, выходящих из этой вершины, отмечены ‘1’ в слайсе *Y*. Слайс *Y* не пуст и в строке 19 выбирается позиция дуги, по которой продолжается обход, остальные дуги заносятся в таблицу *LIFO*. В переменную *node* запоминается двоичный код головы этой дуги. Возможны два случая:

1. из вершины *node* достижимо *k*–1 вершин;
2. из вершины *node* достижимо меньше вершин.

В первом случае процедура продолжает работу в точности также, как если бы проводился поиск в глубину в подграфе, у которого множество вершин равнялось *V\{root}*, начиная с вершины *node*. При возвращении по стеку при *lif* = 2 в цикле (11–16) слайс *Y* останется пустым и процедура закончит свою работу. Дерево состоит из дуги (*root, node*) и дерева поиска в глубину с корнем в вершине *node*.

Во втором случае мы можем при возвращении по стеку при *lif* = 2 в цикле (11–16) слайс *Y* не пуст, поэтому процедура продолжает свою

работу таким же образом, как если бы рассматривался подграф с множеством вершин $V \setminus \{$вершины, достижимые из *node*$\}$. В этом подграфе вершин, достижимых из *root* меньше либо равно k–1, поэтому, по предположению индукции, процедура строит дерево поиска в глубину для этого подграфа. Дерево поиска в глубину для исходного графа равно объединению двух полученных поддеревьев.

Так как посещаются только достижимые вершины, и только один раз, обработка одной вершины выполняется за время пропорциональное log$n$ и возврат происходит по древесным ребрам, то время работы процедуры равно $O(n_r \cdot \log n)$.■

Заключение

В работе представлен ассоциативный алгоритм, выполняющий поиск в глубину в ориентированном графе, и обоснована его корректность. Отметим, что этот алгоритм можно использовать также и для поиска в глубину в неориентированном графе. Для этого каждое ребро *(u, v)* надо записать в список дуг как <*u, v*> и <*v, u*>. В этом случае алгоритм также будет выполняться за время $O(n_r \cdot \log n)$, так как время его работы не зависит от количества дуг.

Заметим, что время работы ассоциативного алгоритма меньше времени работы последовательного (*O(n+m)* для связанного графа), так как все дуги, заходящие в пронумерованную вершину, удаляются за один такт. Разница в множителе log$n$ появляется из-за того, что операция MATCH, соответствующая проверке на равенство, выполняется не за один такт, как это обычно считается на последовательной ЭВМ, а за log n.

Литература

1. Tarjan R. Depth –First Seatch and Linear Graph Algorithms. SIAM Journal of Computing, 1(2): 1972, 146-160.
2. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы построение и анализ. МЦНМО Москва 2000.
3. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. М.: Вильямс, 2000.
4. Kumar V., Rao V.N. Scalable Parallel Formulations of Depth-First Search, V. Kumar, P. S. Gopalakrishnan, L. N. Kanal (eds.), Parallel Algorithms for Machine Intelligence and Vision, Springer, 1990, 1-41.

5. Reinelfeld, V. Schnecke, Work-Load Balancing in Highly Parallel Depth-First Search, Proceedings Scalable High Perfomance Computing Conference SHPCC'94, Knoxville, Te, IEEE Comp. Sc. Press, 1994, 773-780.
6. Nepomniaschaya S. Language STAR for Associative and Parallel Computation with Vertical Data Processing, Proc. of the Intern. Conf. «Parallel Computing Technologies», World Scientific, Singapure, 1991, 258-265.
7. Фостер К. Ассоциативные параллельные процессоры. М: Энергоиздат, 1981.
8. Nepomniaschaya S., Dvoskina M.A. A Simple Implementation of Dijkstra's Shortest Path Algorithm on Associative Parallel Processors, Fundamenta Informaticae, IOS Press, Amsterdam, v. 43, 2000, 227-243.
9. Nepomniaschaya S. Efficient implementation of Edmonds' algorithm for finding optimum branchings on associative parallel processors, Proc. of the Eighth Intern. Conf. on Parallel and Distributed Systems (ICPADS'01). – KyongJu City, Korea: IEEE Computer Society Press, 2001, 3-8.

ДЕКОМПОЗИЦИЯ ПОСЛЕДОВАТЕЛЬНЫХ ПРОГРАММ ПРИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЯХ

Е.С. Борисов

Институт Кибернетики им. В.М.Глушкова НАН Украины
eborisov@ukrpost.net

1. Введение

Для параллельных вычислительных систем необходимо создавать специальные программы. В тексте такой программы определяются части (ветки), которые могут выполнятся параллельно, а также алгоритм их взаимодействия. Можно выделить два основных подхода к построению параллельных программ [1] :

- **распараллеливание по управлению**. Вычислительная задача разбивается на несколько относительно самостоятельных подзадач и каждый процессор загружается своей собственной подзадачей. (соответствует архитектуре MIMD)

- **распараллеливание по данным**. Весь исходный массив данных разбивается на подмножества и фрагменты распределяются между процессорами системы. Одни и те же операции выполняются одновременно над несколькими элементами массива данных. (соответствует архитектуре SIMD)

Параллельные программы можно писать "вручную", непосредственно вставляя в нужные места вызовы коммуникационной библиотеки. Этот путь требует от программиста специальной подготовки. Альтернативой является использование систем автоматического и полуавтоматического распараллеливания последовательных программ.

Автоматические системы распараллеливания [5] выполняют декомпозицию последовательного алгоритма самостоятельно. На вход подается последовательная программа, на выход выдается её параллельный аналог. Пример такой системы это BERT77 средство автоматического распараллеливания Fortran-программ [6].

В случае **полуавтоматической системы распараллеливания**, в тексте последовательной программы выделяются блоки, которые могут выполнятся параллельно. Обычно, в текст вставляются специального вида комментарии, которые игнорируются обычным (последовательным) компилятором. Примером такой полуавтоматической системы может служить Adaptor – одна из реализаций спецификации HPF (High Performance Fortran)[4].

В данной статье предлагается полуавтоматическая система распараллеливания программ на C++ для популярной коммуникационной библиотеки MPICH.

2. Полуавтоматическая система декомпозиции

Предлагаемый препроцессор языка C++ построен в рамках парадигмы распараллеливания по данным. В текст последовательной программы помещаются комментарии специального вида (обычный C++ компилятор их игнорирует), заменяемые препроцессором на соответствующие MPI-вызовы.

Определим метки препроцессора:

- `$INIT$` – инициализация MPI
- `$FIN$` – завершение работы MPI

- `$ROOT$` и `$ENDROOT$` – код между этими метками выполняется только одним (нулевым) процессом
- `$SUM$ DECOMP(i=m,n) REDUCE(sum) REDUCETYPE(type)` и `$ENDSUM$` – распараллеливание подсчета суммы $sum = \sum_{i=m}^{n} f(i)$, где $f(i)$ – алгоритм подсчета, описанный между `$SUM$` и `$ENDSUM$`

Препроцессор можно получит на http://mechanoid.narod.ru

3. Пример

Рассмотрим классический пример параллельного программирования – вычисление π. Число π будем вычислять как определенный интеграл :

$$\int_0^1 \frac{4}{1+x^2}\,dx = 4 \cdot \mathrm{arctg}(x)\,|_0^1 = \pi$$

Согласно правилу прямоугольников интеграл можно заменить суммой:

$$\pi \approx h \cdot \sum_{i=1}^{n} \left(\frac{4}{1+x_i^2} \right) \; ; \; h = \frac{1}{n} \; ; \; x_i = \left(i - \frac{1}{2} \right) \cdot h$$

3.1. Последовательная программа с комментариями

```
#include <iostream.h>
#include <math.h>

int main( int argc, char *argv[])
{
  int i,n = 1000000000;
  double pi,h,x;

  // $INIT$
  h = 1.0 / (double) n; pi= 0.0;
  // $SUM$ DECOMP(i=1,n+1) REDUCE(pi) REDUCETYPE(double)
  for(i=1; i<n+1;i++)
  {
      x = h*((double)i-0.5); pi += (4.0/(1.0+x*x));
```

```
  }
  pi = h * pi;
  // $ENDSUM$
  // $ROOT$
  cout<<"pi = "<<pi<<endl;
  // $ENDROOT$
  // $FIN$
  return 0;
}
```

3.2. Параллельная программа – результат препроцессирования

```
#include <mpi.h>
#include <iostream.h>
#include <math.h>

int main( int argc, char *argv[])
{
  int i,n = 1000000000;
  double pi,h,x;
  int PARnp, PARid,PARlen, PARrc;
  char PARname[MPI_MAX_PROCESSOR_NAME];
  if (PARrc= MPI_Init(&argc, &argv))
  {
      cerr<<"Error starting MPI program.
      Terminating."<<endl;
      MPI_Abort(MPI_COMM_WORLD, PARrc);
  }
  MPI_Comm_size(MPI_COMM_WORLD, &PARnp);
  MPI_Comm_rank(MPI_COMM_WORLD, &PARid);
  MPI_Get_processor_name(PARname,&PARlen);
  cout<<"Process "<<PARid<<" on "<<PARname<<endl;

  h = 1.0 / (double) n; pi= 0.0;
  for(i=PARid+1; i<n+1;i+=PARnp)
  {
      x = h*((double)i-0.5); pi +=(4.0/(1.0+x*x));
  }
```

```
  pi = h * pi;

  double PARpi;
  MPI_Reduce(&pi,&PARpi,1,MPI_DOUBLE,MPI_SUM,0,
  MPI_COMM_WORLD);
  pi=PARpi;

  if (PARid == 0) {
      cout<<"pi = "<<pi<<endl;
 }
  MPI_Finalize();
  return 0;
}
```

3.3. Результаты счета

- Последовательная программа
  Количество итераций 1.00.000.000
  Время счета – 108 секунд

```
[ mpibin ]$ ./pi
 pi = 3.1415926535899708
```

- Параллельная программа на двух процессорах
  Количество итераций 1.00.000.000
  Время счета – 57 секунд

```
[ mpibin ]$ mpirun -machinefile machines -np 2 pi
Process 0 on node2.home.net
Process 1 on node1.home.net
pi = 3.1415926535899708
```

Литература

1. Дорошенко А.Е. Математические модели и методы организации высокопроизводительных параллельных вычислений // Киев: Наукова думка, 2000.
2. Коммуникационные библиотеки – http://www.parallel.ru/tech/tech_dev/ifaces.html
3. MPI – http://www.mpi-forum.org

4. HPF – http://www.crpc.rice.edu/HPFF/
5. Средства распознавания параллелизма в алгоритмах – http://www.parallel.ru/tech/tech_dev/auto_par.html
6. BERT77 – http://www.plogic.com/bert.html

МОДЕЛИРОВАНИЕ ДИНАМИКИ ГРАВИТИРУЮЩИХ СИСТЕМ НА МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ

В.А. Вшивков, Е.В. Неупокоев

ИВТ СО РАН, г.Новосибирск, ИВМиМГ СО РАН, г.Новосибирск

Введение

Одними из наиболее интересных проблем вычислительной астрофизики являются исследование нестационарных процессов, происходящих в галактиках, изучение спиральных узоров и волн плотности в них, моделирование процессов звездообразования и многие другие.

Несмотря на большое количество работ в этой области, как теоретических, так и вычислительных, до сих пор не существует алгоритмов, позволяющих достаточно точно моделировать эволюцию самогравитирующих систем. Широкое применение и развитие за последние несколько лет высокопроизводительных вычислительных систем дает возможность проведения численных экспериментов на основе предлагаемой математической модели.

Цель настоящей работы заключается в создании параллельной программы вычислений для моделирования динамики рассматриваемых самогравитирующих систем с высоким пространственным разрешением.

1. Математическая модель задачи

Образование структуры диска представляет собой задачу многих тел в самосогласованном гравитационном поле. Хорошим приближением для моделирования самогравитирующего диска является бесстолкновительное кинетическое уравнение Власова-Лиувилля:

$$\frac{df}{dt} = \frac{\partial f}{\partial t} + \vec{V}\frac{df}{d\vec{r}} + \frac{\vec{F}}{m}\frac{\partial f}{\partial \vec{V}} = 0\,, \qquad (1)$$

где $f(t,\vec{r},\vec{V})$ есть функция распределения частиц по координатам и скоростям. Вместе с уравнением Пуассона для самосогласованного гравитационного поля

$$\Delta\Phi = 4\pi G\rho\,, \qquad (2)$$

где в правой части ρ суть средняя плотность вещества, распределенного в пространстве, G – гравитационная постоянная, они образуют систему уравнений звездной динамики [1]. При этом значения гравитационной силы F в (1) определяются как градиент потенциала Φ.

Задача решается в трехмерной области в цилиндрической системе координат. На границах всей области для уравнения Пуассона задаются граничные условия первого рода. Вдоль направления угловой координаты φ задаются периодические граничные условия. В качестве модельного примера в центре расчетной области помещается плоский диск радиуса R_0. Уравнения (1)–(2) вместе с граничными условиями для гравитационного потенциала и начальными условиями для частиц составляют математическую модель задачи.

2. Методы решения уравнений модели

Для решения кинетического уравнения Власова (1) используется метод частиц-в-ячейках (Particle-in-Cell). Согласно этому методу все вещество представляется в виде множества отдельных макрочастиц, перемещение которых определяет эволюцию всей системы [2]. Все макрочастицы находятся внутри замкнутого пространства моделирования, которое разбивается на подобласти, называемые ячейками. Траектория каждой частицы является характеристикой уравнения Власова. Эти траектории описываются уравнениями движения частиц:

$$\frac{d\vec{V}}{dt} = -\nabla\Phi\,; \qquad \frac{d\vec{r}}{dt} = \vec{V}\,. \qquad (3)$$

Решение уравнений (3) осуществляется по схеме Бориса [2]. В качестве дискретного аналога уравнений движения используется разностная схема с перешагиванием второго порядка (leapfrog).

После пересчета координат и скоростей с помощью обратной линейной интерполяции в узлах сетки по частицам определяется массовая плотность вещества Используя найденные значения распределения плотности ρ в узлах сетки, из уравнения Пуассона

вычисляется гравитационный потенциал Ф.Из разностного аналога уравнения для силы на смещенной сетке определяются значения трех компонент гравитационной силы. Далее полученные значения сил интерполируются в местоположение каждой частицы и после этого можно определить новые координаты и скорости частиц, перейдя на новый временной шаг. Число временных шагов выбирается из физических соображений эксперимента.

Основную трудность в задаче моделирования эволюции протопланетного диска составляет решение уравнения Пуассона. Рассматриваемая задача является нестационарной, вследствие чего мы можем использовать посчитанные значения потенциала с предыдущего шага по времени. Именно поэтому одним из эффективных методов решения в применении к данной задаче показал себя метод последовательной верхней релаксации, реализованный в комбинации со схемой прогонки вдоль измерения Z [3,4].

3. Параллельная программная реализация

Для проведения численных экспериментов на компьютере была реализована параллельная программа на языке Си++ с использованием библиотеки передачи сообщений MPI++. Выбор языка обосновывается наличием таких его преимуществ, как переносимость, наличие возможности применения объектно-ориентированного подхода, надежность создаваемого кода, которая обеспечивается, в частности, средствами мощной библиотеки STL со стандартными типами данных и процедурами работы с ними.

Алгоритм, реализующий метод частиц в ячейках, обладает высокой степенью внутреннего параллелизма, так как частицы непосредственно не взаимодействуют друг с другом и могут обрабатываться независимо. Несмотря на это, существуют определенные трудности для эффективной параллельной реализации.

Одной из главных проблем, возникающих при параллельной реализации метода PIC, встает вопрос о методе разбиения области моделирования на части и распределении элементов массивов сеточных переменных и модельных частиц между процессорами.

В качестве стратегии разбиения области была выбрана Эйлерова декомпозиция, при которой массивы сеточных переменных и частицы распределяются на части по числу задействованных процессоров, и каждая частица на протяжении всего процесса моделирования

хранится в локальной памяти того процессора, в котором содержится соответствующая этой частице часть пространства моделирования.

В виду аксиальной симметрии задачи и характера поведения частиц, которые движутся вдоль направления угловой координаты, используемое подмножество процессоров организуется в топологию кольца. В момент инициализации вся область разрезается вдоль угловой переменной на слои, которые распределяются по процессорам. Поскольку в качестве начальной конфигурации рассматривается осесимметричный диск, то при таком разбиении частицы также распределены равномерно по процессорам.

Минимизация пересылок является практически основной задачей при разработке параллельного алгоритма. При выбранном способе разбиения области и методе решения уравнения Пуассона необходимы следующие пересылки. Во-первых, это пересылка частиц из одного сегмента области, расположенного в одном процессоре, в сегмент соседнего процессора, если соседняя ячейка, в которую перелетела частица, располагается в памяти соседнего процессора. Во-вторых, между соседними процессорами нужно производить обмены насчитанными значениями потенциала на пограничных слоях, по которым производилось разбиение области. Все описанные обмены необходимы на каждом шаге по времени.

Кроме этого, пересылки, которые могут потребоваться не на каждом временном шаге, состоят в перемещении нескольких слоев одного сегмента и частиц, принадлежащих этим слоям, соседнему процессору в случае дисбаланса загрузки между процессорами по времени счета.

4. Численные результаты

Был проведен тестовый расчет с начальными условиями, описанными в первой части настоящей работы, на различном числе процессоров. Расчеты проводились на многопроцессорных системах МВС1000М в МСЦ г. Москвы (768 доступных процессоров) и в ССКЦ г. Новосибирска (18 доступных процессоров). Данные системы архитектурно представляют собой множество двухпроцессорных узлов (процессоры Alpha), объединенных сетью Myrinet. Каждый узел имеет доступ к оперативной памяти в 2 Гбайта.

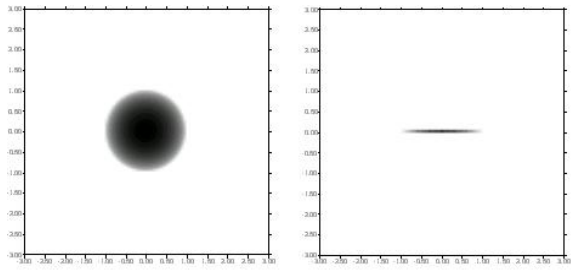


Рис. 1. Проекции средней плотности моделируемой системы на плоскости XY (слева) и XZ (справа). Начальные условия

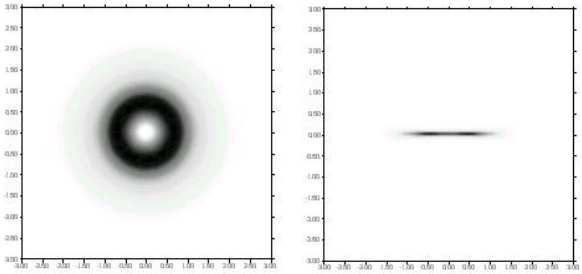


Рис. 2. Проекции средней плотности моделируемой системы на плоскости XY (слева) и XZ (справа) после одного оборота системы

Следует отметить, что основной целью проведения тестового расчета на данном этапе было не исследование физических зависимостей и характеристик, а выявление параметров применяемой программы моделирования и предлагаемой математической модели.

Для приводимого расчета использовались сетка с числом узлов $r \times \varphi \times z = 120 \times 120 \times 240$, и было задействовано 10^7 модельных частиц. Все размеры отнормированы на начальный радиус моделируемого диска, равного $R_0 = 1$. При этом максимальный физический радиус области – $R_{max} = 3$, высота области по z направлению $2*Z_{max} = 6$.

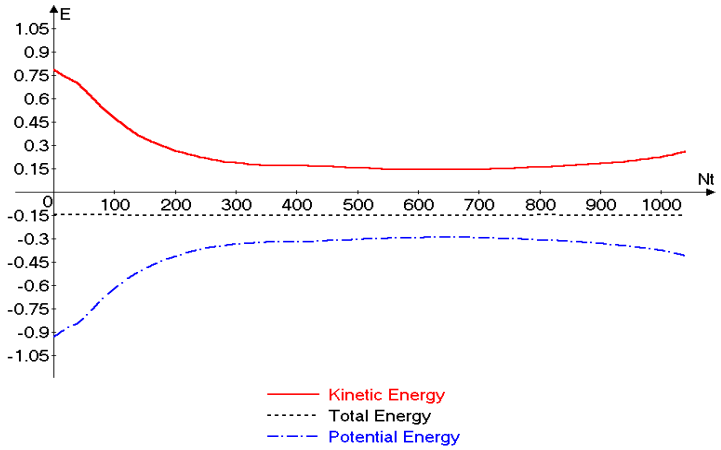


Рис. 3. Кинетическая, полная и потенциальная (сверху вниз) энергии системы.

Характерной единицей времени для рассматриваемой задачи является оборот гравитирующей системы вокруг оси. На рисунках 1 и 2 показаны проекции средней плотности на плоскости XY (слева) и XZ (справа) в начальный момент времени на первом и в момент времени после одного оборота на втором рисунке соответственно для тестового расчета. Проекции представлены в декартовых координатах, размер области 6×6. Можно обратить внимание на то, что диск остается устойчивым в направлении оси OZ.

Одним из основных критериев правильности проводимых расчетов и корректности модели является учет выполнения законов сохранения, в частности выполнение закона сохранения полной энергии. На рисунке 3 приведены кинетическая, потенциальная и полная энергии системы. Как легко видеть, полная энергия системы остается постоянной на протяжении всего времени моделирования.

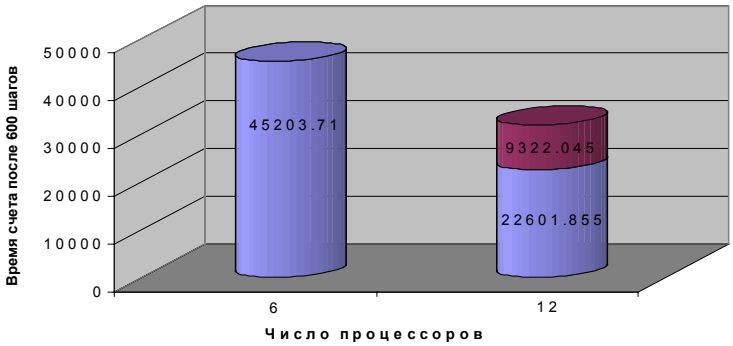


Рис. 4. Эффективность параллельного алгоритма

Для рассматриваемого теста расчеты проводились на 6-ти и 12-ти процессорах. Число частиц, передаваемых из одного процессора в соседний в местах разделения области, составляет порядка 10–20 тысяч. При этом размер передаваемого сечения (одного слоя сетки) при решении уравнения Пуассона составлял $120 \times 240 = 28800$ значений.

В зависимости от шага по времени решение уравнения Пуассона занимает различное время (в него входят как время вычисления на сетке, так и время на передачи сечений на каждой итерации), но каждый раз оно составляет значительную долю времени на каждом шаге, от 32 до 97 процентов всего вычислительного времени на шаг.

На рисунке 4 показано полное время счета программы за 600 временных шагов для одной задачи на различном числе процессоров. Левый столбик соответствует времени счета на шести процессорах. Нижняя часть правого столбика показывает время, которое соответствовало бы идеальному распараллеливанию алгоритма (время счета, меньшее в 2 раза). Весь столбик отражает реальное время счета, достигая значения коэффициента эффективности 1,42.

К этому однако следует добавить, что разработанный алгоритм обеспечивает равномерное разделение данных (как сеточных значений, так и частиц) по процессорам, что легко позволяет масштабировать задачу, увеличивая размерность сетки и используя необходимое число частиц.

5. Заключение

Основным результатом данной работы является реализация параллельной программы моделирования сложной физико-математической задачи. Предложенная математическая модель (1)–(2) реализована с помощью метода частиц-в-ячейках для решения бесстолкновительного уравнения Власова-Лиувилля и с помощью метода последовательной верхней релаксации с прогонкой вдоль измерения *z* для решения уравнения Пуассона.

Проведенные численные эксперименты показывают выполнение основных законов сохранения физики и эффективность разработанного программного обеспечения. Реализованный параллельный алгоритм решения нестационарной задачи в трехмерной области делает возможным исследование начальных стадий джинсовской неустойчивости в трехмерных самогравитирующих системах. Использование разработанной программы позволяет осуществлять моделирование рассматриваемой проблемы на подробных сетках, проводя численные расчеты на многопроцессорных высокопроизводительных компьютерах.

6. Благодарности

Авторы выражают свою благодарность В.Э. Малышкину и В.Н. Снытникову за постановку задачи и полезные обсуждения. Работа была выполнена при поддержке Интеграционных проектов СО РАН 2003 г. №2, №148, и гранта РФФИ 02-01-00864.

Литература

1. Снытников В.Н., Вшивков В.А., Дудникова Г.И., Никитин С.А., Пармон В.Н., Снытников А.В. Численное моделирование гравитационных систем многих тел с газом. // Новосибирск: Вычислительные технологии, 2002, Т 7, № 3.
2. Березин Ю.Б., Вшивков В.А. Метод частиц в динамике разреженной плазмы // Новосибирск: Наука, 1980.
3. Неупокоев Е.В., Тарнавский Г.А., Вшивков В.А. Распараллеливание алгоритмов прогонки: целевые вычислительные эксперименты. // Автометрия, № 4, том 38, 2002, стр. 74-87.

4. Неупокоев Е.В. Математическое моделирование эволюции звездных систем. // Труды молодежной научной конференции, Новосибирск, 2002, стр. 104-109.

АРТ – СИСТЕМА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ НА ОСНОВЕ ЯЗЫКА С ДИНАМИЧЕСКИМИ МАССИВАМИ. ПОДСИСТЕМА ПОДДЕРЖКИ ВРЕМЕНИ ИСПОЛНЕНИЯ

М.А. Городничев

Новосибирский государственный технический университет

Введение

Языки с динамическими типами данных, такие как языки логического программирования, языки сценариев, языки математического программирования, хорошо известны в информатике (например, [7–12]). Объекты динамических типов автоматически размещаются в памяти в момент первого обращения к ним, что освобождает прикладного программиста от забот, связанных с распределением памяти.

Эффективное использование современных вычислительных комплексов невозможно без детального знания архитектуры и методов параллельного программирования. Это ставит новые преграды перед пользователями. На помощь снова могут придти динамические типы, понимаемые уже в более широком смысле как типы, реализация которых берет на себя задачу распределения ресурсов вычислительной системы.

Универсальная поддержка динамических типов не может быть эффективной в смысле затрат времени. Однако существуют широкие предметные области, в программных решениях задач из которых применяются однообразные структуры данных. Такие структуры имело бы смысл реализовать специальными динамическими типами.

Актуальной является задача создания средства автоматизированной разработки параллельных программ, специализированного в области численных методов. Общей структурой данных здесь являются массивы, и распараллеливание

численных алгоритмов зачастую сводится к распределению обработки массивов по узлам вычислительной системы (распараллеливание по данным) [1, 2]. Поэтому очевидный путь создания такого средства – это разработка языка программирования с динамическими массивами.

Предлагаемый язык APT – это расширение языка C++ описателями динамических массивов и редукционных переменных, а также несколькими ключевыми словами, служащими для более структурированного представления программы.

Аналогичный подход используют, например, такие системы параллельного программирования, как HPF [3], DVM [4] и OpenMP [5]. Они также предлагают распараллеливание на основе распределения обработки массивов. В отличие от них система программирования APT является более узко специализированной, предполагая, что обработка массивов производится одним из регулярных способов, характерных для современных численных методов. Вследствие этого APT способна обеспечивать лучшую реализацию динамических свойств параллельных программ, в частности, обеспечить динамическую балансировку загрузки процессоров вычислительной системы.

Система программирования APT

Сценарий применения APT таков: пользователь разрабатывает или берет готовую последовательную программу, добавляет в нее указания для APT, передает результат своего труда компилятору APT и получает на выходе параллельную программу на языке C++, использующую библиотеку системы поддержки вычислений времени исполнения (далее, кратко, – диспетчер).

Ключевые слова разметки структуры программы выделяют явно такие ее части как инициализация вычислений (включая ввод данных), вычисления и обработка результатов (вывод данных). Последовательные программы, в которых нельзя четко разграничить эти части, должны быть модифицированы. Это ограничение на последовательные программы не является чрезмерным, поскольку приведенный способ построения вычислительной программы естественен, и программисту не составит труда немного помочь системе распараллеливания.

Указания второго типа говорят АРТ о том, как обращаться с данными программы. Основой распараллеливания является распределение обработки массивов. Решать, какие массивы распределять, а какие нет, должен пользователь. Для того чтобы сделать массив распределяемым, пользователь сопровождает обычное в C++ объявление массива специальным ключевым словом. Проанализировав зависимости по данным и предыдущий опыт распараллеливания, пользователь решает, поперек каких размерностей допустимо разрезать массив при распараллеливании, и сообщает это системе при объявлении массива. Размеры всего массива (т.е. такого, каким он был бы в последовательной программе) определяются, вообще говоря, во время исполнения программы.

Чаще всего обработка массивов производиться в циклах, и именно на вычисления в циклах тратится большая часть времени выполнения программы. Анализ циклов позволяет компилятору определить взаимосвязи между массивами. Например, в цикле

```
for( i = 2; i<N; i++ )
A[i] = B[i+1]*C[i-2];
```

i-ый элемент массива A используется вместе с (*i*+1)-ым элементом B и (*i*–2)-ым элементом C. Допустим, все три массива объявлены распределяемыми. Тогда разрезать их надо так, чтобы используемые одновременно (т.е. в одной итерации цикла) элементы массивов располагались на одном вычислительном узле. Это минимизирует время коммуникаций в процессе вычислений. Безусловно, не всегда все так хорошо и ясно, как в приведенном примере, где при правильном распределении массивов обмена данными между процессорами вообще не будет.

Еще одной проблемой, связанной с распределяемыми массивами, является передача массивов для обработки в процедуру. Если текст процедуры доступен для обработки компилятору АРТ, распараллеливание производится по общей схеме. Если же это не так, то должно быть гарантировано, что результат применения процедуры отдельно на каждом процессоре к его части массива будет эквивалентен результату применения процедуры ко всему массиву при последовательном исполнении. Это не выполняется, когда процедура может обращаться к элементам массива, потенциально

расположенным на других узлах и когда результат обработки элемента массива зависит от абсолютных[1] индексов этого элемента. В качестве аргументов процедур могут безболезненно использоваться отдельные элементы распределяемых массивов и неразрезаемые слои или столбцы.

Другой вид объектов, требующий внимания пользователя – это редукционные переменные, т.е. такие переменные, которые в конце выполнения цикла по массиву должны содержать в себе значения суммы элементов массива, максимума среди элементов или результата какой-либо другой статистической операции. Такие переменные тоже должны быть явно определены в программе как редукционные с указанием типа редукции.

Диспетчер

Параллельная вычислительная программа должна динамически настраиваться на размеры задачи, характер данных и доступные ресурсы, поэтому ключевую роль в распределении вычислений играет диспетчер. При распределении вычислений преследуются две цели: равномерность загрузки процессоров и минимизация передачи данных по сети как во время перераспределения загрузки, так и во время счета.

Распределение вычислений распадается на две задачи: первоначальное распределение массива при его создании и динамическая балансировка.

Простейшим способом первоначального распределения является разрезание массива поперек размерности на равные части, но такой способ не всегда хорош. Например, в методе частиц массив описывает пространство моделирования, каждая ячейка массива содержит некоторое количество частиц. Частицы при начальном состоянии могут быть сосредоточены в какой-либо области пространства. В этом случае при раздаче узлам компьютера равных частей массива получим чрезвычайно неравномерную загрузку. Таким образом, пользователь должен помочь системе в первоначальном распределении, сообщая структуру загрузки.

---

[1] Абсолютные индексы элемента задают его положение во всем массиве, а не относительно начала части массива, расположенного на данном процессорном элементе.

Требование минимизации коммуникаций обязывает при динамическом перераспределении сохранять структуру взаимодействия ветвей параллельной программы регулярной. Необходимо также поддерживать размеры блоков достаточно крупными, чтобы объем служебной информации находился в разумном соотношении с объемом данных задачи. При этом принципиально невозможным оказывается достижение абсолютно равномерного распределения загрузки.

При планировании распределения вычислений может весьма помочь статистическая информация, собранная во время предыдущих запусков данной параллельной программы, поэтому предполагается включение в подсистему исполнения модуля сбора и обработки статистической информации.

В связи с распределением вычислений, возникает задача обмена данными между ветвями параллельной программы. Передачи происходят при вводе-выводе простых (нераспределяемых) объектов, при обработке редукций, а также в трех случаях для элементов распределяемых массивов:

- передача слоев, вычисляемых в одном узле ВС и требуемых зависимостями по данным в другом;
- ввода/вывод, производимый на выделенном узле, но заполняющий массивы, распределенные по всей ВС;
- перераспределение вычислений.

Система АРТ предполагает гибкую обработку пересылок распределяемых массивов, об осуществлении которой в большинстве случаев пользователю системы заботиться не приходится. В наипростейшем варианте пересылается блок последовательно хранящихся байтов массива. Пользователь системы АРТ, однако, может определить тип данных, не позволяющий использовать побитное копирование. Например, список, в котором каждый элемент содержит указатель на следующий элемент. В простых случаях компилятор, а в более сложных − программист, определяют метод преобразования объекта в плоское (последовательное) представление, необходимое для передачи состояния объекта по сети, и процедуру конструирования объекта из его плоского представления.

Третьей задачей диспетчера является распределение памяти в узлах вычислительной системы.

При распределении массивов возникают две проблемы, связанных с памятью: как размещать элементы массива на узле ВС и как размещать вспомогательные структуры подсистемы поддержки. В решении этих вопросов необходимо обеспечить, во-первых, правильное создание и уничтожение динамических объектов и, во-вторых, эффективность использования оборудования, т.е. учет архитектурных особенностей современных вычислительных средств.

Для размещения элементов массива в памяти предпочтительным является последовательное их расположение, так как в этом случае обеспечивается наиболее эффективный доступ к данным. Кроме того, нужно предусмотреть использование уже существующих процедур для обработки массивов. Исходный код таких процедур может быть недоступен компилятору, а объектный код процедур ожидает последовательного расположения элементов. В качестве примера служебных структур диспетчера могут быть упомянуты так называемые виртуальные слои, обеспечивающие доступ к удаленным элементам массива. Балансировка вычислений будет приводить к тому, что слои массивов будут перемещаться между узлами, при этом возникает задача реорганизации памяти.

В системе предусматривается механизм отслеживания динамических объектов и их своевременного уничтожения. В качестве элементов динамического массива могут использоваться указатели на объекты некоторого типа, размещаемые в куче, кроме того, элементами могут быть и объекты, агрегирующие другие объекты через указатели. Эта тема уже затрагивалась при обсуждении коммуникаций. Другой вопрос, который возникает вследствие возможности использования указателей на объекты в элементах динамических массивов – отслеживание существующих ссылок. Так, пользователь может переместить указатель на некоторый объект из кучи из одной ячейки динамического массива в другую. С точки зрения программиста, который работает в виртуальной среде, объект остается жить, и не требуется ничего создавать и ничего удалять. Реально же, в системе может произойти передача соответствующего объекта с одного узла ВС на другой. При этом на целевом узле объект должен быть порожден в куче, а в исходном, где он больше не требуется, память из-под него – освобождена.

Современное оборудование и системное обеспечение заставляют задумываться о взаимном расположении в памяти совместно используемых объектов. Неудачное распределение памяти может в несколько раз уменьшать производительность программ. Это ясно в теории и наглядно продемонстрировано в ходе экспериментов, проводившихся на кафедре ПВТ НГТУ. Проблемы связаны с разбиением памяти на банки, особенностями кэширования, виртуальной памятью. Например, последовательные обращения к памяти происходят быстрее, если адресуются рядом расположенные ячейки. Еще пример: в одну и ту же строчку кэша проецируются несколько расположений в памяти. Может получиться, что совместно используемые объекты будут размещены так, что обращение к одному из них будет вытеснять из кэша другой.

Текущее состояние проекта и дальнейшая работа

На текущий момент разработан интерфейс библиотеки диспетчера, идет ее реализация. Построена система моделирования работы алгоритмов балансировки, с ее помощью изучены различные подходы к балансировке (централизованные, локальные алгоритмы) и возможные способы реализации модуля сбора и анализа статистической информации о ходе выполнения параллельных программ.

Сейчас все усилия направлены на завершение реализации первой версии библиотеки. Испытание первой версии должно выявить недочеты в проектировании и указать направления усовершенствования системы. Первой серьезной задачей, на которой планируется проверить новую систему программирования, является задача моделирования гравитационной динамики многих тел методом частиц в ячейках [2,6].

Литература

1. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. Наука, Новосибирск, 1988.
2. Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. In the International Journal on Future Generation Computer Systems (Elsevier). Vol. 17, issue 6, p.755-765, 2001.

3. High Performance Fortran. http://www.crpc.rice.edu/HPFF/
4. DVM система. http://dserv.keldysh.ru/dvm/
5. OpenMP C and C++ Application Program Interface. Version 2.0 March 2002. http://www.openmp.org
6. Вшивков В.А., Малышкин В.Э., Снытников А.В., Снытников В.Н. Численное моделирование гравитационной динамики многих тел методом частиц в ячейках: параллельная реализация // Сибирский журнал вычислительной математики, № 1, 2003, стр. 25-36.
7. Prolog Programming Language http://www.engin.umd.umich.edu/CIS/course.des/cis400/prolog/prolog.html
8. The Lisp Programming Language http://www.engin.umd.umich.edu/CIS/course.des/cis400/lisp/lisp.html
9. Perl Programming. http://www.perl.com
10. Python Language Web Site. http://www.python.org
11. Matlab. http://www.mathworks.com/matlabcentral/
12. MatCAD. http://www.mathsoft.com/

РЕАЛИЗАЦИЯ МЕТОДА ОКРЕСТНОСТЕЙ В ЗАДАЧАХ РАСПОЗНАВАНИЯ ОБРАЗОВ С ИСПОЛЬЗОВАНИЕМ XML WEB SERVICES И КЛАСТЕРНЫХ СИСТЕМ

В.П. Гергель, А.В. Гришагин

Нижегородский государственный университет им.Н.И.Лобачевского

Введение

В данной статье рассматриваются некоторые аспекты задачи распознавания образов и одного из подходов к их решению, называемого «методом окрестностей», а также способ реализации метода в виде XML Web Services как шлюза к кластерной системе.

Под задачами распознавания образов понимаются проблемы отнесения объектов классификации к тем или иным ранее указанным множествам – классам. При этом информация о классах дается в виде совокупности объектов, для которых распределение по классам уже известно.

Существует большое количество подходов к построению алгоритмов распознавания, и многие из этих подходов требуют для

решения дополнительной информации типа разделимости классов гиперповерхностями. Новый метод в задачах распознавания образов, разработанный в Нижегородском государственном университете им. Н.И. Лобачевского, называется «*метод окрестностей*». Этот метод обладает рядом преимуществ по сравнению с другими существующими подходами.

Метод окрестностей

Метод окрестностей обеспечивает вычисление обобщенных оценок близости, подобных оценкам типа «ближайшего соседа», без сопоставления с каждым элементом обучающего набора и не требует задания разделяющих поверхностей.

Основная идея метода состоит в следующем. Во множестве возможных изменений значений параметров объектов вводится система окрестностей, и близость любой пары точек характеризуется минимальной охватывающей их окрестностью из этой системы. С системой окрестностей связана эффективная процедура, позволяющая непосредственно определить точки обучающего набора, лежащие в любой выбранной окрестности, без сортировки всех точек этого набора, т.е. определение ближайших соседей распознаваемого объекта не требует сопоставления его со всеми элементами обучающего набора, и, таким образом, увеличение обучающего набора не оказывает существенного влияния на время распознавания. Кроме того, метод окрестностей оказывается эффективным в задачах, где количество признаков объектов оказывается достаточно большим.

Основу метода составляют множественные непрерывные отображения многомерных областей на одномерные шкалы (типа классических *кривых Пеано*), что и обеспечивает указанные преимущества, а так же принцип *прецедентности*, согласно которому решение принимается путем подбора «похожих» элементов из обучающего набора.

Полное описание метода можно найти в [1].

Реализация метода окрестностей в виде XML Web Services

В качестве средства реализации «метода окрестностей» в задаче распознавания образов была выбрана технология .NET XML Web Services, предлагаемая корпорацией Microsoft как новый подход к

написанию клиент-серверных приложений. Веб-Сервис (Web Service) – особый вид серверного приложения, реализующий логику приложения и предоставляющий клиенту интерфейс для вызова т.н. веб-методов с целью выполнения необходимых операций.

Разработанный комплекс «Платформа Принятия Решений .NET» состоит из Веб-Сервиса, реализующего метод, и клиентского приложения, взаимодействующего с этим сервисом.

Клиентская часть «Платформы Принятия Решений .NET» реализована в виде Интернет-приложения с использованием ASP.NET WebForms, и, таким образом, любой пользователь, без установки дополнительного программного обеспечения, с использованием обычного браузера, может создавать и решать задачи распознавания образов. Использование технологии Microsoft XML Web Services позволит в будущем крайне просто включать метод окрестностей в новые проекты, без какой-либо переработки взаимодействия клиентов и сервиса.

Использование кластерных систем

Основу метода окрестностей составляют множественные развертки типа кривых Пеано, и данное обстоятельство позволяет достаточно легко реализовать параллельный вариант алгоритма. Под множественными развертками понимается, что для каждой точки рассматриваемой области необходимо построить сдвинутый образ, а затем каждый из них отобразить на одномерную шкалу. Количество сдвинутых образов определяется необходимой точностью дискретизации непрерывных величин.

Развертки вычисляются независимо друг от друга, что позволяет применить достаточно простое распараллеливание с очень большим приростом производительности. Ускорение в данном случае можно получить фактически равным числу доступных процессоров. Однако надо отметить тот факт, что при небольшой величине дискретного приближения может оказаться эффективнее использовать один процессор, т.к. развертки вычисляются достаточно быстро и накладные расходы на передачу данных (использование MPI) могут перевесить выигрыш от параллельных вычислений. Но в тех случаях, когда количество признаков объекта велико и требуется высокая степень приближения (в этом случае придется использовать

арифметику неограниченной точности), параллельные вычисления оказываются лучшим решением.

Для реализации предложенного подхода идет работа по изменению существующего Веб-Сервиса распознавания образов таким образом, чтобы имелась возможность обращаться к кластерной системе через этот сервис. При этом предоставляемые сервисом веб-методы останутся прозрачными для разработки клиентских приложений, но внутренняя логика Веб-Сервиса будет переработана с учетом параллельных вычислений разверток типа Пеано на кластере.

В качестве решения проблемы коммутации между сервисом и кластерной установкой возможна интеграция с системами управления заданиями на кластере, разрабатываемыми в Нижегородском государственном университете ([3], [4], С. 184-187, С. 78-82).

Помимо использования существующих проектов управления кластером идет разработка пилотной версии системы управления кластерной установкой в виде нового XML Веб-Сервиса. Такой подход позволит не только осуществлять взаимодействие через специализированное клиентское программное обеспечение, но и через собственные пользовательские приложения.

Литература

1. Гергель В.П., Стронгин Л.Г., Стронгин Р.Г. Метод окрестностей в задачах распознавания, Известия АН СССР. Техническая кибернетика, №4, 1987.
2. Горелик А.Л. Гуревич И.Б. Скрипкин В.А. Современное состояние проблемы распознавания. М.: Радио и связь, 1985.
3. Лабутин Д.Ю. Система удаленного доступа к вычислительному кластеру (менеджер доступа): Высокопроизводительные параллельные вычисления на кластерных системах. Материалы второго международного научно-практического семинара, Нижний Новгород: Издательство Нижегородского университета, 2002. С.184-187
4. Гергель В.П., Свистунов А.Н. Разработка интегрированной среды высокопроизводительных вычислений для кластера Нижегородского университета: Высокопроизводительные параллельные вычисления на кластерных системах. Материалы второго международного научно-практического семинара,

Нижний Новгород: Издательство Нижегородского университета, 2002. С.78-82.
5. Дж. Просиз. Программирование для Microsoft NET. М.: Издательско-торговый дом «Русская редакция», 2003.
6. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие, Нижний Новгород: Издательство Нижегородского университета, 2001.
7. Общий обзор систем управления кластером http://nhse.cs.rice.edu/NHSEreview/CMS/

ИНСТРУМЕНТАЛЬНАЯ СИСТЕМА РАСПРЕДЕЛЕННОЙ ГЛОБАЛЬНОЙ ОПТИМИЗАЦИИ

В.А.Гришагин, Я.Г.Ленц

Нижегородский государственный университет им.Н.И.Лобачевского

Программная система предназначена для решения следующей базовой задачи: Пусть $\varphi(x)$ есть действительная функция, определённая в гиперкубе D N-мерного евклидова пространства, т.е.

$$D = \left\{ x \in R^n : a_i \le x_i \le b_i, 1 \le i \le N \right\},$$

где a, $b \in R^n$ есть заданные векторы, и пусть в точке x^* (предполагается, что такая точка существует) функция $\varphi(x)$ достигает минимального значения на множестве D, т.е.

$$\varphi(x^*) = \min_{x \in D} \varphi(x).$$

Требуется построить оценку $x_* \in D$ точки x^* (для некоторого принятого понятия близости, например, чтобы $||x^* - x_*|| < \varepsilon$, где ε есть заданная точность) на основе конечного числа k значений функции, последовательно вычисленных в выбранных точках области D.

Трудность численного решения сформулированной задачи связана с тем, что искомая точка x^* является интегральной характеристикой функции $\varphi(x)$в области D, так как для отождествления некоторой точки с точкой наименьшего значения необходимо сопоставить значение функции в этой точке со значениями функции во всех остальных точках области. Число необходимых испытаний при увеличении

размерности области D увеличивается экспоненциально. Для решения поставленной задачи используем два подхода [1,2]: многошаговую схему редукции размерности и редукцию размерности при помощи разверток. При размерности области D порядка 10 в разумные сроки решить задачу на одном ПК не возможно. Для этого необходимо объединить вычислительные ресурсы нескольких машин. Процесс поиска глобального минимума может происходить одновременно на нескольких вычислительных узлах.

Представляется возможным создать систему, объединяющую небольшое число (сотни) персональных компьютеров в единый вычислительный механизм по решению поставленной задачи.

Рассматривая систему в первом приближении можно выделить основные требования:

1. Отсутствие ограничений на размерность задачи.
2. Наличие удобного инструмента постановки задачи.
3. Возможность выбора подхода к решению.
4. Независимость от числа ПК.
5. Надежность с возможностью круглосуточной работы семь дней в неделю.

Основой для будущей системы целесообразно бы было выбрать клиент-серверную схему. В данном случае на сервере должны существовать инструменты создания задачи, инструменты управления решением, механизмы сбора статистики и инструменты работы с ней, механизмы работы с клиентами. Таким образом, основной задачей сервера будет снабжение клиентов задачами и сбор от них данных по их решению. На клиента же возлагается работа по решению поставленной сервером задачи и передаче результатов. Далее и будем рассматривать подобную схему. При этом клиента можно вооружить либо общей схемой решения многомерной задачи, либо схемой решения одномерной задачи. Более подходящим представляется первый вариант, т.к. в этом случае возможно более гибкое распределение подзадач между клиентами, а кроме того, при дальнейшей эволюции системы представляется возможным переход к многоуровневой схеме распределения подзадач, при которой клиенты могут, в свою очередь, являться серверами для клиентов более низкого уровня. В качестве метода решения одномерных задач используем

характеристическую схему. При этом имеет смысл не нагружать клиента конкретными алгоритмами, а сделать способным реализовывать переданные сервером методы в рамках характеристической схемы [3].

Рассмотрим поочередно предложенные требования к системе:

1. Отсутствие ограничений на размерность задачи.

В полном объеме это требование выполнить не удастся в виду конечности машинных ресурсов, но реализовать систему, способную работать с задачей в несколько сотен аргументов возможно.

2. Наличие удобного инструмента постановки задачи.

Чтобы поставить задачу для системы необходимо задать а) исследуемую функцию б) область поиска в) точность г) методы решения. Если говорить о задаче размерностью несколько сотен аргументов, то целесообразно создать отдельный элемент системы для конструирования подобного рода функций. Приложение, отвечающее за подготовку функции, в своей основе должно содержать подобие редактора формул. Также необходимо обеспечить пользователя широким набором математических функций и возможностью использования заранее созданных функций вновь. Возможно, имеет смысл реализовать способность вычислять значение функции и строить различные сечения. Задача в целом выполнимая, но существует одна проблема – область определения. Заданная функция должна быть определена и непрерывна в области поиска. Необходимо будет решить проблему проверки области определения. Выбор способа задания области поиска тоже требует проработки. При большой размерности задачи табличное задание гиперпараллелепипеда может быть неудобным. Возможно, имеет смысл разработать язык описания задачи и компилятор. При этом исследуемая функция тоже может быть представлена в терминах этого языка в купе со своей областью определения.

3. Выбор подхода к решению

Задачу многомерной оптимизации можно решить путем редукции размерности. Для этого можно использовать либо многошаговую схему, либо редукцию при помощи разверток. На данном этапе реализация этих схем очевидных проблем не имеет.

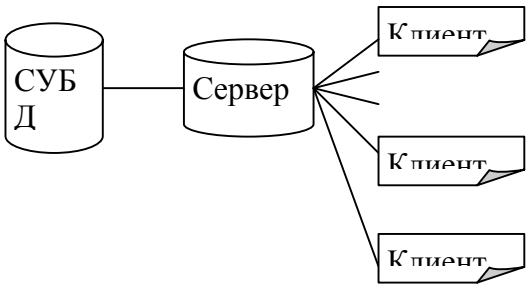
4–5. Независимость от числа ПК, надежность.

Требуемого уровня надежности, универсальности и масштабируемости можно добиться путем применения одной из технологий построения распределенных систем. В настоящий момент наибольший интерес вызывают технологии CORBA и COM (COM+, DCOM). Обе технологии обеспечивают примерно одинаковый и достаточно высокий уровень абстракции. Выбор между этими технологиями зависит от многих параметров. Немаловажным аспектом является выбор платформы. Если не ориентироваться целиком на Windows, то CORBA будет предпочтительнее. Но в отличие от бесплатной COM, CORBA – коммерческая технология.

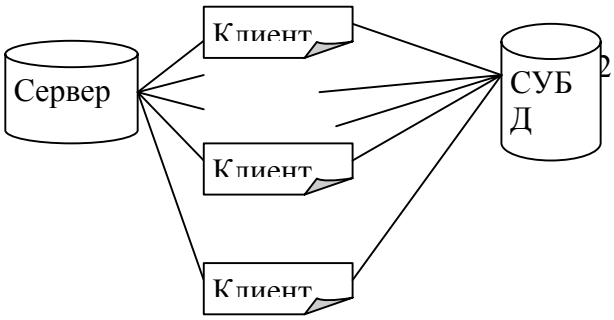
Отдельного обсуждения требуют вопросы сбора и обработки статистики. Схема сбора и обработки статистики должна реализовывать самое полное протоколирование процесса вычисления. Но в этом случае возрастает объем информации, пересылаемый по сети. Если информацию о каждом испытании возвращать на сервер, то это скажется и на быстродействии. Предлагается несколько вариантов решения этой проблемы: а) Предоставить пользователю возможность регулировать объем собираемой статистики в процессе решения б) Статистику можно передавать не в процессе вычислений, а единовременно вместе с результатом вычислений клиентом в) Проблемой сбора статистики и работы с ней можно озадачить СУБД.

В последнем случае можно предложить следующие схемы взаимодействия :

1.



2.



В преимуществах первого подхода – простота клиента. Взаимодействие клиента только с «родным» сервером. Нет необходимости установки на клиентских машинах «клиента» СУБД и не возникает вопросов с организацией прав доступа клиентов на СУБД.

При втором подходе удается разгрузить «сервер», но взамен придется организовывать распределение клиентов СУБД по клиентским вычислительным узлам. Что может повлечь за собой дополнительные финансовые затраты.

Также многое зависит от степени интеграции инструмента работы со статистикой и серверной части. Если инструмент будет выполнен в качестве отдельного программного модуля, то сложности подключения его к вычислительному серверу и серверу СУБД равнозначны. Но если полностью изолировать серверную часть вычислительной системы и инструменты работы со статистикой не удастся, то придется решать вопрос об интеграции инструмента с вычислительным сервером, либо о разделении инструмента на два, один из которых будет частью сервера, другой – отдельным модулем, работающим с СУБД.

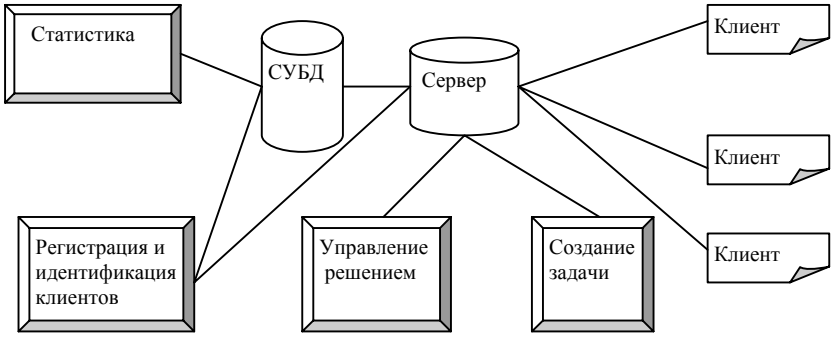
В качестве статистики необходимо учитывать количество испытаний по каждой координате, номер испытания, на котором найдено минимальное значение. В случае задач большой размерности, когда вычисление каждого значения требует значительных ресурсов, полезно было бы сохранять вычисленные значения для дальнейшего использования.

Значительное число «клиентов» также требует наличия инструмента, с помощью которого можно во-первых учитывать клиентов, во-вторых идентифицировать их при повторном подключении. Это позволит видеть статистику работы по каждому клиенту. Логично было бы реализовать такую службу регистрации и идентификации, используя СУБД.

Чтобы система могла решать задачи размерностью в сотни переменных, необходимо обеспечить возможность подключения сотен клиентов. Для этого необходимо, чтобы система выходила за рамки одной локальной сети. Таким образом, серверная часть должна жить в

интернете. В этом случае имеет смысл строить систему так, чтобы клиенты «общались» только со своим вычислительным сервером.

Поэтому на этапе проектирования разумно остановиться на следующей схеме системы:



Литература

1. Стронгин Р.Г. Численные методы в многоэкстремальных задачах. Информационно- статистический подход. М.: Наука, 1978.
2. Strongin R.G., Sergeyev Ya.D. Global optimization with non-convex constraints: Sequential and parallel algorithms, Kluwer Academic Publishers, Dordrecht, Netherlands. 2000.
3. Strongin R.G., Sergeyev Ya.D., Grishagin V.A. Parallel Characteristical Algorithms for Solving Problems of Global Optimization // Journal of Global Optimization,10, 1997. P. 185–206.
4. Sergeyev Ya.D., Grishagin V.A. Parallel asynchronous global search and the nested optimization scheme, Journal of Computational Analysis & Applications, 3(2), 2001. P. 123–145.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ СПЕЦКУРСА «ПРОГРАММИРОВАНИЕ ДЛЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ»

А.Г. Деменев

Пермский государственный университет
Пермский государственный педагогический университет

Введение

Для студентов магистратуры факультета информатики и экономики Пермского государственного педагогического университета автором доклада разработан и читается с 2000 года спецкурс «Программирование для параллельных вычислительных систем». Данный курс предназначен для студентов обучающихся по программе «Информатика в образовании», получающих степень магистра и квалификацию «Учитель информатики, преподаватель высшей школы».

Основные цели и задачи курса:

- знакомство с основами параллельных вычислений;
- углубление образования в области информатики;
- развитие практических навыков в компьютерном моделировании, алгоритмизации и программировании.

Данный спецкурс включает в себя как лекционную часть, так и лабораторный практикум. В лекционной части курса даются основные теоретические понятия и примеры ПВС, излагаются типичные приемы и методы программирования для таких систем.

Чрезвычайно важную роль в курсе играют лабораторные работы. При их выполнении предусматривается следующие режимы (один из них или сочетание – по выбору преподавателя): разработка и отладка программы для ПВС и проведение по ней расчетов; выполнение расчетов «вручную» с заполнением всех необходимых таблиц для промежуточных результатов; проведение расчетов в среде, имитирующей ПВС.

Для выполнения работ №4-5 требуется доступ к высокопроизводительному кластеру. В результате этих лабораторных команде студентов из 2-х или 3-х нужно выполнить комплексное задание разработать параллельное приложение с использованием коммуникационной библиотеки MPI. Методологии программирования

иллюстрируются на сравнительно несложных задачах, например, требующих реализации отдельных алгоритмов из области линейной алгебры, вычисления определенных интегралов и т.д. Так как разработка параллельных программ практического уровня сложности представляет собой многоэтапный технологический процесс, то, к сожалению, она не может быть продемонстрирована во всей полноте на таких задачах.

Общие указания к лабораторным работам № 1–3

Лабораторные работы № 1–3 представляют собой микроисследования, использующие методы математического моделирования и имитационного моделирования. Отчет по работе должен быть представлен в виде твердой копии и в электронной форме.

Для выполнения требуется PC с установленным программным обеспечением: пакет Multi-Pascal 2.0 для DOS или для Windows, табличный и текстовый процессор (например, входящие в состав MS Office или OpenOffice). Используются материалы главы 5 ученого пособия автора [1]. Общие рекомендации студентам для выполнения лабораторных работ расположены на стр.50 вышеуказанного пособия.

Простейшая модель, описывающая ускорение (коэффициент эффективности распараллеливания) S, которое может быть получено на компьютере из N процессоров, может быть получена из закона Амдала:

$$S \le \frac{1}{f + (1-f)/N}$$

где f – доля участков в программе, которые не могут быть распараллелены. Крайние случаи в значениях f соответствуют полностью параллельным (f=0) и полностью последовательным (f = 1) программам. Используя профилировщик (или отладчик), можно оценить время выполнения на однопроцессорной машине последовательного участка программного кода (t) и всей программы (T). Тогда $f = t/T$ и

$$S_{\max} = \frac{1}{t/T + (1-t/T)/N}$$

Реальная выгода от увеличения числа процессоров всегда меньше:

- в системах с разделяемой памятью – в основном из-за ожидания процессоров возможности получения доступа к памяти;
- в системах с распределенной памятью – в основном из-за задержек межпроцессорных коммуникаций;
- во всех параллельных системах – из-за накладных расходов на порождение параллельных потоков.

При анализе результатов, необходимо сравнивать результаты компьютерного моделирования в среде Multi-Pascal с предсказаниями такой простой математической модели. Где возможно, подбирать более реалистичную эмпирическую модель, включающую дополнительно 1-2 подгоночных параметра. Рекомендуется при построении модели использовать теоретические оценки коммуникационной трудоемкости параллельных алгоритмов, которые для ряда типичных случаев можно найти в учебном пособии [2].

Лабораторная работа №1 «Знакомство с имитатором ПВС – средой Multi-Pascal»

Задания к работе приведены в главе №5 вышеуказанного пособия [1]. Для выполнения работы ориентировочно требуется 4 ч в лаборатории с преподавателем и 4 ч самостоятельной работы. Лабораторная работа предназначена для знакомства со средой Multi-Pascal. Студент действует строго в соответствии с инстукциями. Ему дается готовая программа – вычисление числа Пи. Он должен сам определить: что делает программа, и для какой целевой платформы она написана. В этом ему помогают комментарии, встроенные в программный код. Студент должен найти в программе параллельные и последовательные части кода, установить между разными частями контрольные точки. Запуская программу на выполнение с заданными параметрами, он должен оценить время выполнения на однопроцессорной машине последовательного участка программного кода (t) и всей программы (T).

Затем должен получить таблицу оценок ускорения имитационой модели S и по формуле Амдала S_{max} для разного числа процессоров. Построив по данным, записанным в электронной таблице, графики функций $S = S(N)$ и $S_{max} = S_{max}(N)$ в логарифмическом масштабе, студент должен сравнить их между собой и сделать выводы. Требуется оценить среднее время порождения одного процесса τ, считая его

подгоночным параметром в полуэмпирической формуле для ускорения:

$$S(N) = \frac{1}{N\tau/T + t/T + (1 - t/T)/N}.$$

Студент должен понимать, из каких предположений была получена эта формула.

Лабораторная работа №2 «Моделирование многопроцессорных систем с разделяемой памятью в Multi-Pascal»

Задания к работе можно взять из главы №5 пособия [1]. Для выполнения работы ориентировочно требуется 6 ч в лаборатории с преподавателем и 6 ч самостоятельной работы. Первое задание – моделирование вычислений на многопроцессорных компьютерах монопольного доступа. Требуется изучить зависимость коэффициента эффективности распараллеливания от N, проанализировать результаты и сделать выводы. Варианты заданий формируются требованием: использовать разные программы типовых алгоритмов (вычисления числа Pi, перемножения матриц, метода Jacobi и т.п.) и/или разные входные параметры (число расчетных точек, размер матриц и т.д.). Число вариантов – в зависимости от имеющихся в наличии готовых программ.

Второе задание – моделирование вычислений на многопроцессорных компьютерах коллективного доступа. Требуется построить таблицы и графики зависимостей математического ожидания и стандартного отклонения коэффициента эффективности распараллеливания от *N*, проанализировать результаты и сделать выводы.

Лабораторная работа №3 «Моделирование многопроцессорных систем с распределенной памятью в Multi-Pascal»

Задания к работе приведены в главе №5 учебного пособия [1]. Для выполнения работы ориентировочно требуется 8 ч в лаборатории с преподавателем и 8 ч самостоятельной работы. Первое задание – моделирование вычислений на многопроцессорных компьютерах монопольного доступа. Требуется промоделировать вычисления на 64-процессорных компьютерах, следующих параллельных архитектур: линейная, кольцо, двумерная решетка, двумерный тор, трехмерная

решетка, гиперкуб, полносвязный граф. Требуется изучить зависимости коэффициента эффективности распараллеливания от архитектуры компьютера, проанализировать результаты и сделать выводы. Второе задание – моделирование вычислений на кластерах монопольного доступа. Делается оно аналогично первому заданию, но устанавливается в десятки раз большая задержка межпроцессорных коммуникаций. Третье задание – моделирование вычислений на многопроцессорных компьютерах заданной топологии. От студента требуется провести моделирование для заданной архитектуры.

Лабораторная работа №4 «Знакомство с базовым ПО кластера рабочих станций и средой программирования на основе MPI»

Работа выполняется на параллельной вычислительной системе типа «кластер рабочих станций», со средой программирования для языка C/C++, имеющей поддержку работы с установленной коммуникационной библиотекой MPI. Задания соответствуют наиболее частым действиям при создании параллельных программ на имеющейся в распоряжении студентов и преподавателя вычислительной платформе. Для выполнения работы ориентировочно требуется 4 ч в лаборатории с преподавателем и 4 ч самостоятельной работы.

Лабораторная работа №5 «Программирование для кластеров рабочих станций»

Работа выполняется на параллельной вычислительной системе типа «кластер рабочих станций», со средой программирования для языка C/C++, имеющей поддержку работы с установленной коммуникационной библиотекой MPI. Задания предусматривают выполнения в рамках команды из 2-3х человек проекта разработки эффективной параллельной реализации алгоритма из области линейной алгебры, вычисления определенных интегралов и т.д. Эффективная реализация подразумевает, что, по возможности, выполняются следующие условия: вычислительная работа в параллельных процессах не дублируется; нагрузка на каждый процесс примерно одинакова; при увеличении числа доступных процессоров в n раз, время решении задачи уменьшается почти в n раз. Для

выполнения работы ориентировочно требуется 8 ч в лаборатории с преподавателем и 8 ч самостоятельной работы.

Обсуждение

Материалы спецкурса прошли успешную апробацию в Пермском госпедуниверсите. Ряд материалов спецкурсов используется при обучении студентов Пермского госуниверситета. Все студенты успешно справляются с первыми четырьмя лабораторными работами практикума, которые выполняются индивидуально. Это позволяет достичь основных целей спецкурса. Пятая заключительная работа под силу лишь тем командам, в составе которых имеется хотя бы один студент с ярко выраженными способностями к программированию. Результаты апробации разработанного лабораторного практикума позволяют рекомендовать использовать его материалы при подготовке специалистов к эффективному использованию современных высокопроизводительных вычислительных систем.

Благодарности

Выражаю признательность проф. Д.В. Любимову, руководителю научной компоненты НОЦ «Неравновесные переходы в сплошных средах» Пермского госуниверситета, за предоставленную поддержку моей работы в рамках научной стажировки при кафедре теоретической физики ПГУ. Рад выразить признательность проф. Е.К. Хеннеру, проректору ПГУ по информатизации и новым технологиям обучения, за внимание к моей работе и постоянную ее поддержку. Благодарю доц. И.И. Вертгейма, с.н.с. Института механики сплошных сред (г.Пермь), за многократные полезные обсуждения разработанных материалов практикума. Работа выполнена при финансовой поддержке Пермского госпедуниверситета (проект «Развитие и применение параллельных методов компьютерного моделирования многомасштабной нелинейной динамики спиновых систем», 2002–2003 гг.). Работа выполнена при частичной финансовой поддержке из средств гранта РЕ-009-0 Американского Фонда Гражданских Исследований и Развития (АФГИР).

Литература

1. Деменев А.Г. Параллельные вычислительные системы: основы программирования и компьютерного моделирования. Учебное пособие к спецкурсу.– Пермь: ПГПУ. 59 с.
2. Гергель В.П., Стронгин Р.Г. Основы параллельных вычислений для многопроцессорных вычислительных систем. Учебное пособие. Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2000. 176 с.

МЕЛКОЗЕРНИСТЫЙ ЛОКАЛЬНО-ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ДЛЯ ЧЕТЫРЕХТОЧЕЧНОЙ НЕЯВНОЙ РАЗНОСТНОЙ СХЕМЫ УРАВНЕНИЯ ТЕПЛОПРОВОДНОСТИ

Г.В. Заручевская (Лозинская), О.В. Севостьянова, О.А. Юфрякова, И.Н. Кожин

Поморский государственный университет, г.Архангельск

Введение

Господствующим способом распараллеливания задач до сих пор является крупноблочное. При этом задача разбивается на большие подзадачи (блоки), предназначенные для параллельного решения на небольшом числе процессоров. Соответственно ориентированы и параллельные алгоритмы численного решения задач.

Очевидно, что с ростом числа процессоров блоки измельчаются и вычисления в подавляющем большинстве случаев будут идти медленнее: параллелизм вырождается. Избежать вырождения можно только при условии, что обмены происходят и одновременно, и локально, т.е. расстояние между взаимодействующими процессорами мало и не зависит от размера задачи. Число процессоров фактически неограниченно и уже в 90-х достигло 100 000.

Итак, для многопроцессорных систем лучше использовать мелкозернистое локально-параллельное программирование (МЛПП). При этом задача должна быть разбита на множество небольших однотипных подзадач, которые будут исполняться параллельно на отдельных вычислительных машинах (ВМ). Данные максимально распределены по системе, а программы в каждой ВМ используют

минимально возможные наборы данных. В общем случае число обменов имеет тот же порядок, что и число вычислительных операций. Таким образом, мелкозернистость, или массовое распараллеливание означает, что в каждом вычислительном процессе в каждый момент времени содержится минимальное число команд и данных.

Есть два обязательных условия, при которых не происходит снижения производительности МЛПП: 1) Локальность взаимодействий, когда обмен данными происходит только в пределах ограниченного физического и структурного радиуса, независимо от размеров задачи и системы. 2) Параллелизм взаимодействий, когда все возможные в данный момент обмены совершаются параллельно и одновременно с процессом счета.

Следует различать два типа операций: локальные и глобальные. Локальные операции используют ограниченную окрестность процессора. Глобальные операции не ограниченны окрестностью ВМ – это, например, запуск, ввод, прерывание, останов, вывод. Количество глобальных операций не должно влиять на оценку временной сложности задачи.

Разработка и исследование МЛПП для задач математической физики – одно из актуальных направлений современного параллельного программирования. В настоящее время для этих целей применяются модели клеточных автоматов. Целью настоящей работы является разработка МЛПП-алгоритмов решения сеточной задачи для неявной разностной схемы уравнения теплопроводности для массово-параллельных процессоров.

Физическая интерпретация задачи

Процесс распространения тепла в одномерном стержне $0<x<l$ описывается уравнением теплопроводности

$$c\rho\frac{\partial u}{\partial t}=\frac{\partial}{\partial x}\left(k\frac{\partial u}{\partial x}\right)+f_0(x,t),$$

где $u=u(x,\ t)$ – температура в точке стержня в момент t, ρ – плотность материала, $c\rho$ – теплоемкость единицы длины, k- коэффициент теплопроводности, f_0 – плотность тепловых источников. В общем случае k, c, ρ, f_0 могут зависеть не только от x и t, но и от температуры $u = u(x,\ t)$ (квазилинейное уравнение теплопроводности) и даже от

$\partial u/\partial x$ (нелинейное уравнение). Если k, c, ρ постоянны, то уравнение теплопроводности можно записать в виде

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2} + \mathrm{f},\ \mathrm{f} = \frac{\mathrm{f}_0}{\mathrm{c}\rho},$$

где $a^2 = k/(c\rho)$- коэффициент температуропроводности. Без ограничения общности можно считать $a = 1$, $l = 1$.

Разностная схема

Разностное уравнение может быть записано в виде:

$$y_{k-1}^{\nu} - (\frac{h^2}{\tau} + 2) y_k^{\nu} + y_{k+1}^{\nu} = -\frac{h^2}{\tau}(y_k^{\nu-1} + \tau \cdot \mathrm{f}_\mathrm{k}^{\nu}). \qquad (1)$$

$k = 1..N–1$, $\nu=1..M$.

Эта разностная схема неявная, для ее решения применяется метод прогонки. Рассмотрим идею решения этой разностной схемы, используемую для распараллеливания задачи.

1. Решаем системы линейных уравнений методом прогонки:

а) решаем M систем из N-1 уравнения каждая

$$z_{k-1}^{\nu} - \left(\frac{h^2}{\tau} + 2\right) z_k^{\nu} + \mathrm{z}_{k+1}^{\nu} = -h^2 \mathrm{f}_\mathrm{k}^{\nu}\ , \qquad (2)$$

где $k = 1..N–1$, ν- номер слоя прямоугольной сетки, $\nu = 1..M$.

б) решаем N–1 систем из N–1 уравнения каждая

$$x_{k-1}^{p} - \left(\frac{h^2}{\tau} + 2\right) x_k^{p} + x_{k+1}^{p} = \begin{cases} 0, & \text{если } k \neq p \\ -\dfrac{h^2}{\tau}, & \text{если } k = p \end{cases} \qquad (3)$$

где $k = 1..N–1$, p – номер процессора, $p = 1..N–1$.

2. Учитывая, что систему линейных алгебраических уравнений (1) можно представить в виде:

$$H \cdot Y^{\nu} = -h^2 \cdot F^{\nu} - (h^2/2) \cdot E \cdot Y^{\nu-1},$$

тогда решение систем линейных уравнений (1) представимо в виде $Y^{\nu} = Z^{\nu} + y_1^{\nu-1} \cdot X^1 + y_1^{\nu-1} \cdot X^2 + y_2^{\nu-1} \cdot X^3 + \ldots + y_{N-1}^{\nu-1} \cdot X^{N-1}$, где Z^{ν}, X^1, X^2, X^3, …, X^{N-1} –решения соответствующих систем (2) и (3).

Замечание. Для решения этих трехдиагональных систем линейных уравнений здесь применяется один из вариантов метода прогонки. Рассмотрим его на примере системы (2) при $\nu = 1$.

Положим $\alpha = h^2/\tau$, $F_i = -h^2 \cdot f_i^1$

I. Из первого и последнего уравнения системы выписываются соотношения между z_1^1 и z_2^1, z_{N-1}^1 и z_{N-2}^1:

$z_0^1-(2+\alpha)z_1^1+z_2^1=F_1 \Rightarrow z_1^1=1/(2+\alpha)z_2^1+(z_0^1-F_1)/(2+\alpha)$; $\aleph_1=1/(2+\alpha)$;

$\wp_1=(z_0^1-F_1)/(2+\alpha)=(z_0^1-F_1)\cdot\aleph_1$;

$z_{N-2}^1-(2+\alpha)z_{N-1}^1+z_N^1=F_1 \Rightarrow z_{N-1}^1=1/(2+\alpha)z_{N-2}^1+(z_N^1-F_{N-1})/(2+\alpha)$;

$\aleph_{N-1}=1/(2+\alpha)=\aleph_1$; $\wp_{N-1}=(z_N^1-F_{N-1})/(2+\alpha)=(z_N^1-F_{N-1})\cdot\aleph_1$.

II. «Прямой ход прогонки» $\aleph_k=1/(2+\alpha-\aleph_{k-1})$; $\wp_k=(\wp_{k-1}-F_1)/(2+\alpha-\wp_{k-1})$, $k=2..N-2$

III. Находим $z^1_{N-1}=(\wp_{N-1}-\aleph_{N-1}\cdot\wp_{N-2})/(1-\aleph_{N-1}\cdot\aleph_{N-2})$;

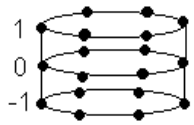
IV. «Обратный ход прогонки» $z^1_k=\aleph_k\cdot z^1_{k+1}+\wp_k$, $k=(N-2)\ldots1$.

Остальные системы линейных уравнений решаются аналогично.

Отметим также, что $\aleph_k$, $k=1..(N-1)$ для всех систем будут одинаковыми.

Мелкозернистый локально-параллельный алгоритм решения сеточной задачи четырехточечной неявной разностной схемы для уравнения теплопроводности

Реализация алгоритма выполняется на цилиндре, в основании которого кольцо из N–1 процессора, а каждая образующая содержит 3 процессора.

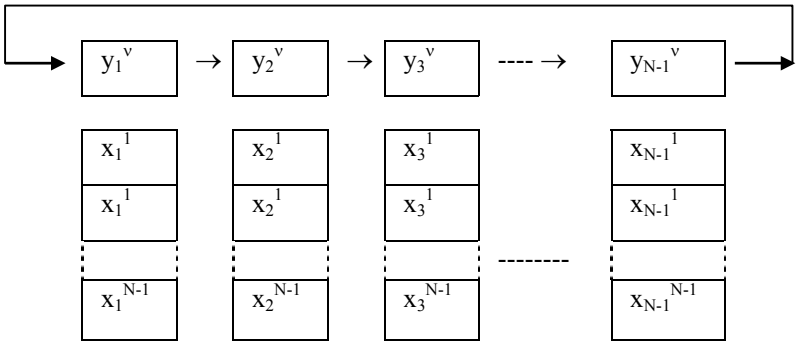


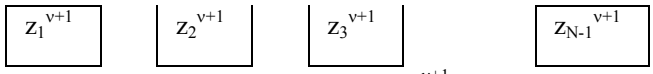
Этап 1. Хост-машина рассчитывает $\aleph_k$, $k=1..(N-1)$. Далее она рассылает эти значения в полном объеме каждому процессор нулевого кольца и покомпонентно в +1 и –1 кольцо. Исходные данные хост-машина рассылает следующим образом: на процессоры кольца +1 загружаются значения $-h^2\cdot f_k^{\nu}$, где ν – нечетное число и одновременно номер соответствующего слоя; на процессоры кольца –1 загружаются значения $-h^2\cdot f_k^{\nu}$, где ν – четное число и одновременно номер соответствующего слоя; в каждый из N–1 процессоров кольца 0 загружаются значения $(-h^2/\tau)\cdot e_i$, где e_i-единичный вектор, i-тая компонента которого равна 1.Там же хост-машина размещает начальные значения y^0_i. Одновременно i является и номером процессора на кольце.

Этап 2. Каждый процессор в кольце с номером 0 согласно методу прогонки находит вектор X^p, где p-номер процессора кольца и номер соответствующего вектора решения системы уравнений (3). Процессоры кольца +1 рассчитывают Z^1, а кольца –1 рассчитывают Z^2, где Z^1 и Z^2 – соответствующие векторы решения системы уравнений (2). В этих кольцах за N сдвигов в одном направлении рассчитываются $\wp_k$, за следующие N сдвигов рассчитываются z_k^{ν}, ν=1 или ν=2 в зависимости от номера кольца, где k – номер процессора в кольце. Каждое значение $\wp_k$ и z_k^{ν} сохраняется в вычислительном модуле за номером k, т. е. в том же процессоре, в котором они вычислены.

Этап 3. Умножение матрицы на вектор выполняется на кольце процессоров с номером 0. Каждый i-й процессор кольца с номером 0 (i=1..(N–1)) в начале выполнения цикла программы содержит, Y_0^{ν} и i-тую строку матрицы X; i-ые элементы векторов Z_k^{ν} процессоры этого кольца получают путем сдвига из колец +1 и –1. Умножение производится за N–1 такт. Каждый такт включает в себя накопление и сдвиг элемента вектора Y^{ν} по регулярному каналу (последний такт – только накопление), причем накапливают только те процессоры, в которых содержится элемент вектора Y^{ν}. Наличие данного элемента фиксируется значением шаблона присутствия в ЭМ (шаблон присутствия сдвигается циклически по структуре вместе с вектором Y^{ν}). По завершению умножения результат суммируется с соответствующей компонентой вектора Z_k, готовый вектор $Y^{\nu+1}$ записывается и пересылается на кольцо.

Структура, на которой производится выполнение программы на третьем этапе выглядит следующим образом:





После расчета элементов вектора Y^{v+1} начинается вычисление элементов вектора Y^{v+2} по тому же алгоритму. Параллельно на кольца x +1 и –1 вычисляются соответственно Z_k^{v+2} и Z_k^{v+3}.По окончанию расчета Z_k^{v+2} и Z_k^{v+3} пересылаются на кольцо 0. Цикл повторяется [(M+1)/2] раз. Вычислительный процесс заканчивается, когда выполнен расчет последнего слоя Y^M.

Алгоритм имеет существенные недостатки. Во-первых, на кольцах +1 и –1 большую часть времени процессоры простаивают. Во-вторых, только третий этап является локальным и мелкозернистым алгоритмом, в то время как второй относится к крупноблочному типу.

ИСПОЛЬЗОВАНИЕ СРЕДСТВ MPI В ПРОГРАММНОЙ СИСТЕМЕ ДЛЯ МОДЕЛИРОВАНИЯ ДИНАМИКИ БОЛЬШИХ АНСАМБЛЕЙ НЕЛИНЕЙНЫХ ОСЦИЛЛЯТОРОВ

М.В. Иванченко, О.И. Канаков, К.Г. Мишагин, В.Д. Шалфеев

Нижегородский государственный университет им. Н.И.Лобачевского

Введение

Исследование свойств динамики дискретных нелинейных сред представляет собой сегодня одно из наиболее перспективных и развивающихся направлений теоретической физики. Математической моделью дискретной среды является система большого количества обыкновенных дифференциальных уравнений (ОДУ), связи между которыми имеют топологию правильной решетки в пространстве одного, двух или большего числа измерений. Такие модели могут быть получены в результате сеточной аппроксимации пространственных дифференциальных операторов в уравнениях распределенных систем. Однако, наибольший интерес представляют пространственно-дискретные модели систем, которые изначально, по своей физической природе, обладают свойством дискретности. Такими являются, например, биологические ткани, имеющие клеточное строение. Исследование динамики сердечной ткани как дискретной возбудимой среды может быть особенно полезно с точки зрения проблем

медицины (возникновение и подавление режима фибрилляции). К дискретным средам также можно отнести кристаллические решетки твердых тел и некоторые искусственные системы: сети синхронизации (например, в фазированных антенных решетках), решетки джозефсоновских контактов, искусственные нейронные сети. Необходимо подчеркнуть, что в решеточных системах могут наблюдаться эффекты, принципиально не реализуемые или структурно-неустойчивые в распределенных системах, например, дискретные бризеры (пространственно-локализованные осциллирующие решения).

Назначение и возможности пакета

Остановимся вкратце на возможностях программного пакета, разработанного авторами [1]. Этот программный комплекс предназначен для моделирования на параллельных вычислительных системах динамики двумерных решеточных ансамблей, в которых каждый элемент связан с четырьмя своими ближайшими соседями. Динамика базового элемента может задаваться отображением или системой обыкновенных дифференциальных уравнений (ОДУ) любого порядка в форме Коши. В уравнениях допускается явная зависимость от времени. Для интегрирования ОДУ используется алгоритм, основанный на разностной схеме Рунге-Кутты четвертого порядка. Однако, пакет допускает «подключение» других алгоритмов счета. Заданные пользователем уравнения динамики базового элемента применяются единообразно ко всем узлам решетки. Граничные условия могут быть типа Неймана, Дирихле или периодическими. Допускается задание произвольного распределения на решетке любого числа параметров, что позволяет моделировать пространственно-неоднородные системы. Начальные условия также могут задаваться произвольным образом.

Пакет содержит средства регистрации осциллограмм динамики отдельных элементов, позволяет делать «снимки» состояния всей решетки в фиксированные моменты времени. Имеется прототип обработчика, позволяющего вычислять значение старшего ляпуновского показателя траектории системы в процессе моделирования. Возможно создание и подключение других обработчиков.

Использование пакета для моделирования динамки нелинейной решеточной среды

Известно, что в цепочечных и решеточных ансамблях нелинейных (консервативных) осцилляторов могут существовать дискретные бризеры [2]. Они представляют собой локализованные в пространстве осциллирующие решения, реализующиеся при определенном классе начальных условий. Крайне интересным поэтому является вопрос о возможности появления бризеров из случайных условий в результате некоторого переходного процесса. Оказывается, что из случайных начальных условий, соответствующих ненулевой температуре решетки возможно образование некоторого количества бризеров, если на границах решетка

В форме Коши уравнения движения системы имеют вид

$$\dot{x}_i = y_i$$
$$\dot{y}_i = -f(x_i) + a(x_{i-1} - 2x_i + x_{i+1})$$
$$i=1, \ldots, N,$$

где $f(x) = x - x^2 + x^3/4$, $a = 0.1$, $N = 3000$.

Известно [1, с. 193], что такая система имеет пространственно-локализованное решение, реализующееся при начальных условиях $x_{1501}(t=0) = 2{,}3456$, $x_{i\neq 1501}(t=0) = y_i(t=0) = 0$. Этот результат был повторен с помощью представленного пакета. На рисунках 1, 2, 3 показаны минимальные и максимальные значения переменных x_i элементов решетки за три различных интервала времени. Как видно, лишь небольшая часть энергии начального возмущения излучается в решетку в течение некоторого начального промежутка времени; в дальнейшем локализованное решение с хорошей точностью сохраняет свою амплитуду.

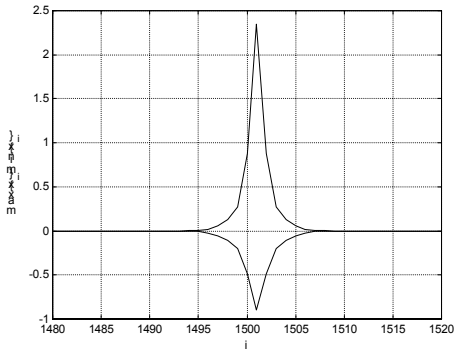


Рис. 1 (t = 0–30)

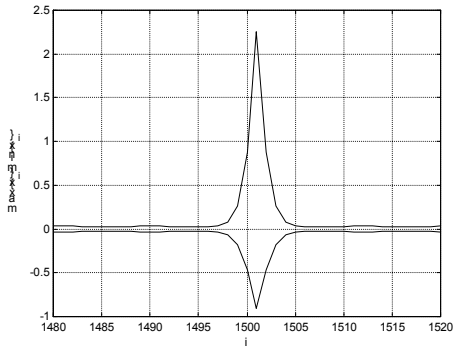


Рис. 2 (t = 230–260)

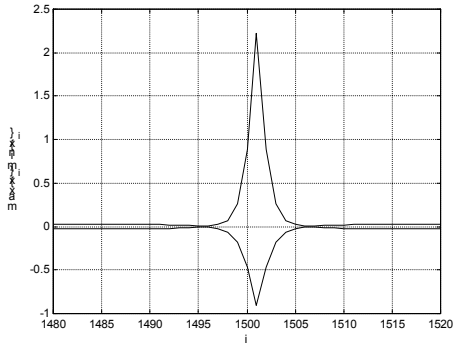


Рис. 3 (t = 260–290)

Заключение

Подводя итог, отметим, что представленный программный продукт, используя преимущества параллельного программирования и будучи переносимым благодаря использованию MPI, является весьма эффективным средством для изучения процессов, происходящих в дискретных активных средах, что на настоящий момент является одной из самых актуальных задач нелинейной динамики, нелинейной физики.

Литература

1. Иванченко М.В., Канаков О.И., Мишагин К.Г., Осипов Г.В., Шалфеев В.Д. Материалы второго Международного научно-практического семинара Высокопроизводительные параллельные вычисления на кластерных системах 26-29 ноября 2002 г.
2. S. Flach, C.R. Willis Discrete Breathers //Physics Reports. 1998. V.295, №5.

ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ОПТИМИЗАЦИИ НА ОСНОВЕ МЕТОДА СЛУЧАЙНОГО ПОИСКА

О.А. Кузенков, А.Л. Ирхина, М. Жулин

Нижегородский государственный университет им. Н.И.Лобачевского

Направление по развитию методов оптимизации, учитывающих специфические особенности разных классов экстремальных задач, в настоящее время является весьма актуальным. Существует уже большое количество разработанных методов по этому направлению, использующих некоторые предположения относительно целевой функции, однако фактический вывод простых алгоритмов осуществлен в большинстве случаев для одномерного случая. На практике же, для решения многомерных задач часто используется подход сведения многомерной задаче к последовательности одномерных (что само по себе может оказаться весьма сложным).

К рассмотрению предлагается метод для решения задачи оптимизации действительной, строго положительной функции $J(z)$,

определенной на некотором компактном множестве Ω конечномерного пространства. В основе рассматриваемого метода лежит хорошо известный алгоритм случайного поиска глобального экстремума (метод Монте-Карло). К настоящему времени авторами разработано несколько модификаций предлагаемого метода, использующихся в различных случаях, в зависимости от особенностей целевой функции экстремальной задачи (большая размерность, большое количество локальных экстремумов или же наоборот – унимодальность, овражистость, и т.д.).

В отличие от простого алгоритма случайного поиска, для получения новой точки предложенные модификации используют информацию о ранее найденных точках. Каждой из полученных точек на каждом шаге ставится в соответствии некоторая величина p_i, являющаяся вероятностью выбора этого решения, как опорного на следующем шаге. и отражающая степень важности учета информации об этом решении. Величина p_i имеет следующий вид

$$p_i = (J(z_i))^m (\sum_{j=1}^{m} (J(z_j))^m)^{-1},$$

где m – количество сделанных шагов.

Новая точка здесь рассматривается, как многомерная случайная величина с плотностью вероятности, которая зависит от выбранной с вероятностью *pi* опорной точки. В различных модификациях плотность вероятности реализована по-разному. На ранних модификациях использовались функция плотности нормального распределения $f(\| x - x_i \|) = \frac{1}{\sqrt{2\pi\sigma}} \exp(\frac{1}{2} \frac{| x - x_i |^2}{\sigma})$ функция $f(\| x - x_i \|) = = \frac{a_i}{\sigma (x - x_i)^2 + \varepsilon}$. В более старших версиях она зависит еще и от номера итерации, с ростом номера шага плотность вероятности выбора новой точки стремится к δ-функции. С течением времени новые точки начинают накапливаться в окрестности глобального максимума. В последней модификации для повышения информативности метода на каждом шаге вычисляется не одна, а несколько новых точек, (их количество задается как параметр метода),

это полезно при большой размерности и вычислительной сложности оптимизируемой функции.

В этом случае целесообразно применять распараллеливание предложенного алгоритма, Основными этапами работы метода являются следующие:

- из множества решений выбирается опорная точка;
- рассчитываются координаты новых точек;
- вычисляются в них значения оптимизируемого функционала;
- вычисляется величина $\sum_{j=1}^{m}(J(z_j))^m$;
- для каждого полученного решения вычисляется вероятность p_i.

За исключением первого пункта техника распараллеливания применима ко всем этапам. Созданная программа, реализующая предложенный алгоритм выполнена с использованием библиотеки параллельного программирования MPI.

О.А. Кузенковым сформулированы доказаны теоремы о сходимости метода, так же им доказано, что этот метод применим и в случае, когда целевая функция определена в бесконечномерном пространстве.

Предлагаемый метод удачно применяется для решения задач с оптимальным управлением (в случаях, когда такая задача может быть сведена к задаче оптимизации), и использовался для решения следующих задач: оптимальное управление процессом охлаждения участка стержня при постоянной внутренней энергии, оптимальное выделение заданной гармоники при сохранении постоянной суммы фазовых координат. Так же он применялся для расчета компонент плоского тензора деформаций полей смещения на основе анализа изменения рельефа поверхности.

PARACON: СИСТЕМА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ НА ТИПОВЫХ АЛГОРИТМИЧЕСКИХ СТРУКТУРАХ

Т.Е. Истомин

Московский государственный университет им. М.В. Ломоносова
istomin@cmc.msu.ru

Введение

Система представляет собой инструментальное средство поддержки высокоуровневого параллельного программирования, основанного на повторном использовании проектных решений и шаблонов алгоритмов (типовых алгоритмических структур). Система содержит библиотеку параллельных реализаций типовых алгоритмических структур и визуальный инструмент – конструктор параллельных программ.

Описание подхода

Типовая алгоритмическая структура (ТАС) – это параметризуемый параллельный алгоритм, шаблон, фиксирующий схему решения некоторого класса задач. Программист использует эту схему как готовое проектное решение при анализе своей задачи, а также использует скелет алгоритма, который предлагается разработчиком данной ТАС, наполняет его функциональностью и настраивает под свои нужды. Различаются структурные и функциональные параметры. Структурные параметры уточняют топологию алгоритма (например, размерности величин), позволяют масштабировать структуру. Функциональные параметры наполняют ТАС вычислительными операциями, необходимыми для решения поставленной задачи. ТАС с подставленными параметрами будем называть *экземпляром* типовой алгоритмической структуры. Экземпляры ТАС могут выступать в роли функциональных параметров для других экземпляров ТАС.

Основными типовыми алгоритмическими структурами являются: независимый параллелизм (*Map*), редукция (*Reduce*), конвейер (*Pipe*), «плантация» (*Farm*), вычисления на сетке (*Mesh*), последовательная композиция (*Comp*) [1, 2, 7].

Программа на ТАС представляет собой дерево вложенности экземпляров ТАС и определяет структуру параллельного алгоритма.

Листьями дерева являются вручную реализованные функции (последовательные или параллельные).

Данный подход предоставляет возможность отделить описание структуры параллельного алгоритма от функциональности, что может быть использовано для осуществления глобальной оптимизации алгоритма [5, 6].

Библиотека TAC

Библиотечная часть содержит сущности, которыми может манипулировать пользователь в системе. Эти сущности содержат:

- класс на целевом языке – готовая к использованию реализация абстракции;
- метаинформация в формате XML – необходимые конструктору знания о компоненте;
- и, при необходимости, графический модуль, подключаемый к конструктору.

Сущности могут быть следующих видов:

- типовая алгоритмическая структура;
- реализация протокола коммуникаций (рассылка, сборка, поток задач...);
- структура данных (массив, изображение...);
- операция над данными (декомпозиция, агрегация...).

Принципы устройства и функционирования

В отличие от некоторых других систем, основывающихся на том же подходе, принцип данной системы – использование заранее заготовленных объектов без какого-либо вмешательства в их внутреннюю структуру. Каждая из доступных пользователю сущностей является полностью законченной, то есть не абстрактной, и может быть использована в программе без изменения. Пользовательский код всегда оформляется в виде новой типовой алгоритмической структуры, основанной на структуре *Seq* (в листьях дерева), а не в виде реализации абстрактного метода класса, как это делается, например в [3, 4].

Программа строится в SPMD модели, то есть на всех процессорах запускается один и тот же код. Каждая из типовых структур получает при инициализации коммуникатор (группу процессоров с автономной нумерацией), в рамках которого будет осуществляться работа данной

структуры и всех подструктур. Коммуникаторы, на которых размещены подструктуры должны содержать подмножества процессоров коммуникатора внешней структуры и обычно не должны пересекаться.

В системе существует ряд объектов, логически распределенных по группе процессоров. К ним, например, относятся экземпляры ТАС и протоколы обмена. Создаются такие распределенные объекты следующим образом. На каждом процессоре некоторой группы создается локальный представитель распределенного объекта, который в зависимости от своего расположения в группе выбирает вариант поведения. Так как программа на всех процессорах выполняется одна, и одновременно на всех процессорах создаются одноименные представители, группа одноименных объектов ведет себя как один распределенный объект.

Каждому объекту доступны только внутренние координаты процессоров области связи (группы процессоров), в которой он оперирует, поэтому физическое расположение объекта не имеет значения.

Общение типовой алгоритмической структуры с родительской структурой осуществляется посредством внешних протоколов, внутриструктурные коммуникации – это забота самой структуры, но некоторые структуры могут параметризовываться так называемым внутренним протоколом. К примеру, структура *Pipe* содержит встроенную реализацию передачи данных со стадии на стадию, а структуре *Map* протокол распределения данных передается при инициализации.

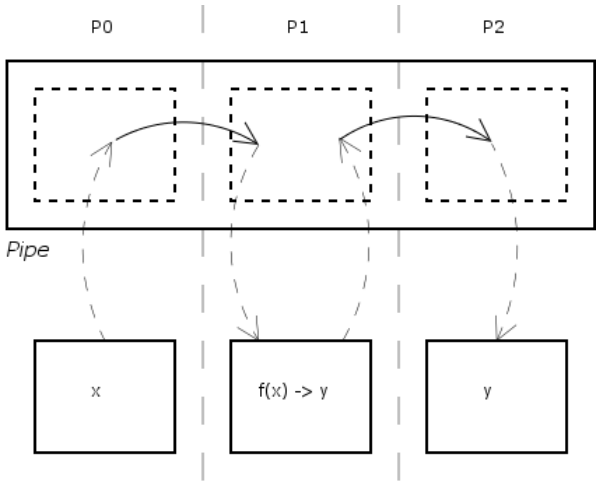
Поддерживаются различные виртуальные топологии групп процессоров – от простой линейки процессоров до многомерной решетки с замыканием по выбранным размерностям.

Модель взаимодействия построена по слоистой схеме. На нижнем, скрытом от пользователя уровне, находится коммуникационная библиотека (например MPI). Следующий уровень – уровень коммуникаторов – создает абстракцию над низкоуровневыми средствами взаимодействия, все структуры реализованы на основе этой абстракции, что обеспечит более легкую переносимость системы и пользовательских программ, а также дополнительных компонентов,

созданных внешними разработчиками. К последнему, верхнему уровню, относятся внутренние и внешние протоколы.

При создании новой ТАС нужно придерживаться некоторых соглашений. Структура должна установить, на какой из ее процессоров надо подавать данные и с какого будут поступать результаты ее работы. Кроме того, если эта структура может иметь вложенные структуры, необходимо реализовать интерфейс внешнего протокола, обеспечить инициализацию и запуск подструктур.

На рисунке представлена схема обменов при организации конвейера с тремя стадиями. Пунктирные вертикальные линии обозначают «границы» процессоров, пунктирные прямоугольники в структуре *Pipe* – представители ее распределенного объекта на каждом из процессоров. Пунктирными дугами обозначены передачи данных между структурами в пределах одного процессора, по внешнему протоколу, сплошными – межпроцессные обмены внутри структуры



Pipe.

Представленный на рисунке конвейер не подключен к какому-либо внешнему протоколу. Первая стадия продуцирует данные *x*, на второй стадии эти данные преобразуются в *y*, третья стадия поглощает результат. Возможно использование конвейера в качестве подструктуры какой-либо другой ТАС, тогда первая стадия сможет

получать данные извне, а последняя стадия – передавать результат наружу.

Процесс исполнения параллельной программы

Выполнение программы состоит из трех стадий. Первая стадия – инициализация объектов, на ней создаются экземпляры всех используемых компонентов (виртуальные топологии, ТАС, коммуникаторы, дистрибьюторы, и т.д.). На этой стадии задаются, в частности, топологии групп процессоров для каждой из типовых структур. Физическое размещение этих групп осуществляется на следующей стадии.

Вторая стадия работы программы – распределение дерева экземпляров ТАС по процессорам. На этой стадии каждая из структур получает коммуникатор, соответствующий топологии структуры, указанной ранее, и распределяет доступные ей процессоры по подструктурам, также в соответствии с их нуждами. Кроме того, на этой стадии представители структур определяют свое местоположение в группе, производят необходимые действия для инициализации структуры как распределенного объекта, определяют какой из подструктур принадлежит данный процессор и выполняют ее инициализацию.

Третья стадия – это, собственно, счет. На всех процессорах выполняется метод `run()` корневой структуры. Она, в свою очередь, вызывает метод `run()` той из своих дочерних структур, множеству используемых процессоров которой принадлежит текущий процессор, затем все повторяется для этой подструктуры и так далее по дереву до листьев.

Конструктор

Конструктор предназначен для параметризации сущностей (создания экземпляров), связывания их между собой с проверкой корректности. После того, как программа собрана в конструкторе, генерируется код с использованием классов из библиотеки системы.

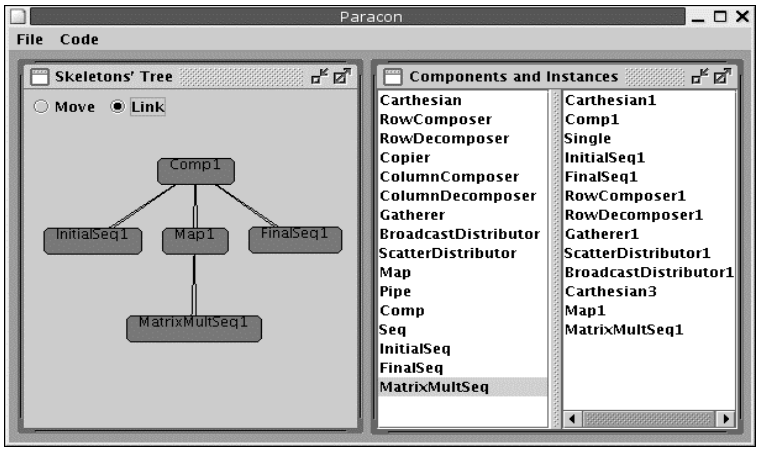
Конструктор предоставляет возможность работы с компонентами в объектно-ориентированном стиле с поддержкой множественного наследования. Определены программные интерфейсы встраиваемых модулей параметризации структур и генерации кода на языках программирования.

Программирование в конструкторе заключается в создании некоторого количества объектов, которые могут ссылаться друг на друга или использовать друг друга.

Экземпляры типовых алгоритмических структур отображаются в окне редактирования дерева, где пользователь может установить связи между ними.

Далее, на этапе генерации кода для каждого созданного в конструкторе объекта в программу вставляется код создания объекта соответствующего класса с подстановкой всех параметров.

На рисунке изображено главное окно конструктора с деревом программы перемножения матриц.



Особенности системы

- объектный подход;
- графическое представление структуры программы;
- возможность использования как стандартных компонентов, так и созданных самостоятельно;
- гибкость встроенных компонентов;
- поддержка произвольной вложенности структур;
- динамическое размещение экземпляров ТАС на процессорах.

Конструктор реализован на языке Java, библиотека – на языке Python с использованием пакета Python MPI.

Литература

1. Берзигияров П.К., Программирование на типовых алгоритмических структурах с массивным параллелизмом. // Вычислительные методы и программирование, 2001, Т.2, 1, с.96-112, http://num-meth.srcc.msu.su/zhurnal/tom_2001/pdf/art2_1.pdf
2. Истомин Т.Е., Система параллельного программирования на типовых алгоритмических структурах. // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы второго Международного научно-практического семинара. / Под ред. Р.Г. Стронгина. Нижний Новгород: Изд-во Нижегородского госуниверситета, 2002.
3. MacDonald S., From Patterns to Frameworks to Parallel Programs, Ph.D. thesis / University of Alberta, Edmonton, Alberta, Canada, 2002.
4. Goswami D., Parallel Architectural Skeletons: Re-Usable Building Blocks for Parallel Applications, Ph.D. thesis / University of Waterloo, Waterloo, Ontario, Canada, 2001.
5. Zavanella A., Skel-BSP: Performance Portability for Parallel Programming.
6. Skillicorn D., Daneluto M., Pelagatti S., Zavanella A., Optimizing Data-Parallel Programs Using the BSP Cost Model.
7. MacDonald S., Parallel Object-Oriented Pattern Catalogue.
8. Coudarcher R., S\'{e}rot J., D\'{e}rutin J., Implementation of a Skeleton-Based Parallel Programming Environment Supporting Arbitrary Nesting.

О СИСТЕМЕ ВЫЯВЛЕНИЯ ПАРАЛЛЕЛИЗМА

А.Ю. Коробко

Таганрогский государственный радиотехнический университет
korobko@mail333.com

Аннотация

В работе рассматривается структура системы выявления параллелизма, предназначенной как для анализа программ, так и для их распараллеливания. Данная система еще не приняла законченного вида, однако уже сейчас может использоваться для достижения поставленных целей [1-4]. Используются следующие алгоритмы: для

анализа зависимостей – алгоритм, реализующий модифицированный омега тест; для распараллеливания – алгоритмы Аллена и Кеннеди, Вольфа и Лам, Дарте и Вивьен. Работа имеет поддержку РФФИ (номера грантов 03-07-06088, 01-07-90211).

Введение

Задача автоматического распараллеливания программ решается на протяжении нескольких десятилетий. За это время было разработанно несколько теорий и большое множество алгоритмов [5]. Также некоторые алгоритмы получили реализацию в виде законченных программных продуктов. Однако данные средства достаточно не развиты и не предоставляют пользователю комплексного подхода, на основе использования различных методов распараллеливания. Таким образом, существующие системы выявления параллелизма носят скорее демонстративный характер, нежели практический.

К тому же данные программные средства были ориентированы зачастую на конечных пользователей, не предоставляя научному сообществу программной среды для анализа параллелизма.

Целью данной работы является разработка модульной программной системы, объединяющей различные подходы к распараллеливанию. Концепция модульности предполагает наличие документированного интерфейса для встраивания дополнительных компонент, реализующих, например, недостающие пользователю алгоритмы распараллеливания.

Таким образом система позволит проводить анализ входной программы, визуализировать данные, осуществлять преобразования над программами и многое другое [6-14].

Данная работа носит ознакомительный характер, демонстрируя назначение и структуру программы.

Структура системы

Система выявления параллелизма состоит из нескольких функциональных модулей (рис. 1).

На первом этапе программа подвергается лексическому и синтаксическому анализу и переводится в промежуточное представление. Промежуточное представление необходимо для удобства выполнения дальнейших этапов с одной стороны и для унификации кода с другой. На данный момент осуществляется

поддержка только языка программирования Fortran, но в дальнейшем предполагается добавление и других распространенных языков программирования. Для этого понадобится лишь добавить соответстующий модуль лексического и синтаксического анализа. Таким образом, внутреннее представление программы представляет входные данные для большинства алгоритмов.

Однако, внутреннее представление помимо самой программы описывает и ее аттрибуты, получаемые на этапах анализа зависимостей, преобразований и др. Таким образом, исключается связь между модулями, и упрощается разработка системы.

Алгоритмы анализа зависимостей являются неотъемлемой частью любой системы распараллеливания. Они дают на выходе информацию, описывающую скрытый параллелизм программы. Эти данные также добавляются во внутреннее представление программы и используются на этапе преобразований.

К другим алгоримам, используемым в системе, можно отнести алгоритмы анализа производительности, выявления определенных участков кода (например, пользователя может интересовать только распараллеливание циклов) и др. Под эту категорию подходят и те алгоритмы, которые добавит пользователь, путем проектирования динамической библиотеки.

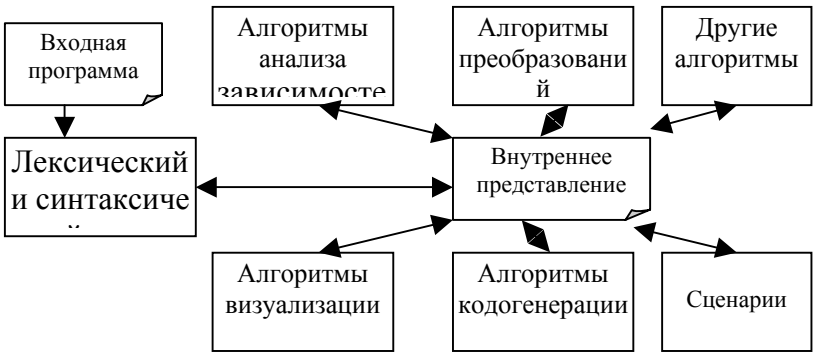


Рис. 1. Структура системы анализа параллелизма

Алгоритмы преобразований на основе внутреннего представления (используется информация о самой программе и ее зависимостях)

позволяют получить параллельный вариант программы. Здесь не ставится задача ориентированности на конечную систему. Таким образом алгоритмы преобразований лишь определяют потенциальные участки кода, которые на некоторой конечной системе могут быть эффективно распараллелены.

Финальным этапом в распараллеливании является этап кодогенерации. Алгоритм кодогенерации на основе промежуточного представления и выбранной пользователем целевой платформы, строит код. В настоящее время используется кодогенерация для распределенной операционной системы для гетерогенных многопроцессорных систем NeurOS [15-17]. Однако в дальнейшем планируется поддержка и таких распространенных систем, как MPI, POSIX Threads и др.

Заключение

Работа представляет обзор структуры системы анализа параллелизма, предназначенной для выполенения комплексного исследования структуры программ. Помимо этого ставится задача распараллеливания и кодогенерации.

В настоящее время реализованно ядро системы, состоящее из модуля управления внутренним представлением, алгоритмов лексического и синтаксического анализа для языка программирования Fortran, алгоритмов анализа зависимостей и преобразований.

В дальнейшем предполагается окончание разработки системы и предоставление ее для свободного доступа.

Литература

1. Коробко А.Ю., Ковтун Н.Г., Транслятор для гетерогенной системы на базе процессора NM6403 // VI Всероссийская научная конференция студентов и аспирантов «Техническая кибернетика, радиоэлектроника и системы управления», Тезисы докладов. – Таганрог: ТРТУ, 2002. – С. 342-343.
2. Коробко А.Ю., Ковтун Н.Г., Бабенко Л.К., Чефранов А.Г., Трансляция программного кода в гетерогенной системе // «Высокопроизводительные параллельные вычисления на кластерных системах» материалы второго Международного научно-практического семинара. – Нижний Новгород: НГУ, 2002. – С. 141-147.

3. Бабенко Л.К., Коробко А.Ю., Чефранов А.Г., Федоров П.А., Макаревич О.Б., О роли компилятора и операционной системы в генерации SPMD задач для гетерогенных систем // СуперЭВМ и многопроцессорные вычислительные системы, Материалы Международной научно-технической конференции. – Таганрог: ТРТУ, 2002. – С. 172-178.
4. Babenko L.K., Chefranov A.G., Korobko A.Yu., Makarevich O.B., Compiler and Operating System Integration for SPMD Tasks Code Generation and Execution in Heterogeneous Cluster Systems // «Informational Systems and Technologies (IST'2002)» Proceedings of the I International conference, Part 3. – Минск: БГУ, 2002. – С. 48-53.
5. Bacon David F., Graham Susan L., and Sharp, Oliver. Compiler Transformations for High-Performance Computing // Computing Surveys. – 1994. – № 26-4.
6. Kelly Wayne, Pugh William. A Framework for Unifying Reordering Transformation: Technical Report. – UMIACS-TR-93-134, CS-TR-3193. – Univ. of Maryland, 1993. – 24 с.
7. Irigoin François. Minimal Data Dependence Abstractions for Loop Transformations: Extended Version // International Journal of Parallel Programming. – 1995. – № 23-4. – С. 359-388.
8. Pugh William, Wonnacott David. Going beyond Integer Programming: Technical Report. – UMIACS-TR-93-132, CS-TR-3191. – Univ. of Maryland, 1992. – 17 с.
9. Darte Alain, Vivien Frédéric. Parallelizing Nested Loops with Approximation of Distance Vectors: a Survey // Parallel Processing Letters. – 1997. – № 7(2). – С. 133-144.
10. Goff G., Ken Kennedy, C-W. Tseng, Practical dependence testing // Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation. – 1991. – № 26-6 – С. 15-29.
11. Darte Alain, Vivien Frédéric. Revisiting the Decomposition of Karp, Miller and Winograd // Parallel Processing Letters. – 1991. – № 3. – С. 45-57.
12. Darte Alain, Vivien Frédéric. On the optimality of Allen and Kennedy's theory for parallelism extraction in nested loops // Journal of Parallel Algorithms and Applications, Special issue on Optimizing Compilers for Parallel Languages. – 1996.

13. Darte Alain, Yves Robert. Scheduling Uniform Loop Nests: Technical Report RR92-10. – Laboratoire de l'Informatique du Parallélisme, 1992.
14. Wolfe Michel. Optimizing Supercompilers for Supercomputers. – Cambridge MA: MIT Press, 1989 – 670 с
15. Бабенко Л.К., Коробко А.Ю., Чефранов А.Г., Федоров П.А., Макаревич О.Б., Распределенная микроядерная архитектура для проектирования операционных систем // СуперЭВМ и многопроцессорные вычислительные системы, Материалы Международной научно-технической конференции. – Таганрог: ТРТУ, 2002. – С. 168-171.
16. Babenko L.K., Chefranov A.G., Fedorov P.A., Korobko A.Yu., Makarevich O.B. Operating System for Multiprocessor System Based on NM6403 Microprocessors // Optical Memory & Neural Networks (Information Optics) – 2002. – № 11-2. – С. 117-121.
17. Babenko L.K., Chefranov A.G., Fedorov P.A., Korobko A.Yu., Makarevich O.B. Mechanisms of Parallel Computing Organization for NeuroCluster // Parallel Computing Technologies: 6th international conference; proceedings / PaCT 2001, Victor Malyshkin (ed.). – 2001. – LNCS 2127. – С. 181-185.

ВИЗУАЛИЗАЦИЯ ДАННЫХ БОЛЬШОГО ОБЪЁМА В РАСПРЕДЕЛЁННЫХ МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ[2]

П.С. Кринов, М.В. Якобовский, С.В. Муравьёв

Институт Математического Моделирования РАН
lira@imamod.ru

Введение

Развитие суперкомпьютерных центров предоставляющих свои ресурсы через интернет, увеличило число научных и прикладных задач решаемых при помощи численного эксперимента. При этом возник новый класс задач. Среди них, проблема визуализации данных большого объёма, который превышает ресурсы персонального компьютера или отдельного вычислительного модуля. Помимо того,

[2] Работа выполнена при поддержке гранта РФФИ №02-01-00589.

отсутствие соответствующих средств визуализации, затрудняет проведение крупномасштабных 3-х мерных расчётов задач газовой динамики, горения, микроэлектроники и ряда других на многопроцессорных системах. Объемы сеточных данных, исходных данных задачи и результатов расчётов, могут превышать размеры оперативной памяти персонального компьютера.

Вычислительная мощность систем растёт очень быстро, увеличивается число процессоров в многопроцессорных системах и таким образом можно ожидать увеличения объёмов результатов вычислений. Наша работа направлена на обработку более сотен гигабайт данных, что требует создания новых методой обработки информации.

Такие объёмы данных не могут разместиться ни в оперативной памяти персонального компьютера, ни даже на жёстком диске. Помимо этого, пропускные способности всех сетей (особенно глобальных) ограничены. Эти ограничения не позволяют воспользоваться стандартными средствами, так как передача данных по любой сети для анализа и визуализации не позволит интерактивно исследовать результаты.

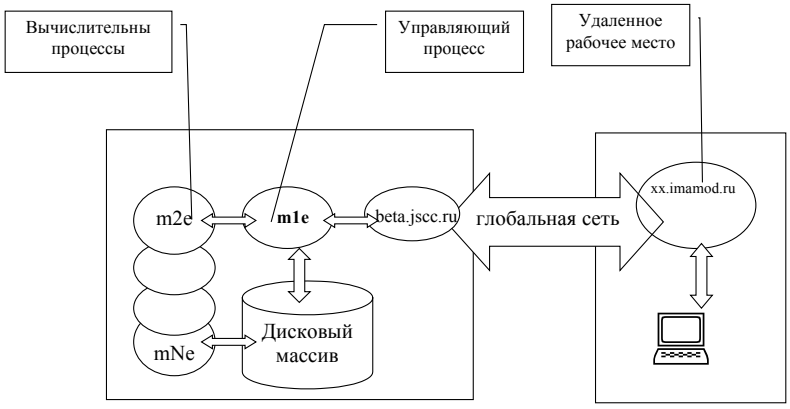


Рис. 1. Структура системы визуализации

Разработанная система визуализации данных большого объема состоит из двух частей – сервера и клиента (модель клиент/сервер, рис.1). Такое разделение позволяет производить основную часть

процесса визуализации на суперкомпьютере и передавать на рабочее место пользователя только информацию, необходимую для построения окончательной картинки. Такой подход предполагает что изображение окончательно формируется на персональном компьютере и таким образом становится возможным использовать современное мультимедийное оборудование (шлемы виртуальной реальности, стерео очки, 3-х мерные манипуляторы и т.д., которым на сегодняшний день оснащено большинство ПК) для большей наглядности информации.

Из существующего множества методов визуализации 3-х мерных скалярных данных выбран метод изоповерхностей – это такие поверхности, на которых рассматриваемая функция (температура, плотность, концентрация и т.д.) принимает фиксированное значение. Таким образом, несколько поверхностей (на которых значения функции выбрано пользователем) вырезаются из 3-х мерного скалярного поля и отображаются на экране.

При таком подходе, некоторая треугольная сетка определяет каждую изоповерхность. Размеры этих сеток в некоторых случаях вполне сравнимы с размерами исходной 3-х мерной сетки и таким образом не могут передаваться по сети за незначительное время для интерактивной визуализации. Более того, эти данные не могут разместиться в оперативной памяти одного процессорного модуля. Таким образом, главной проблемой становиться сжатие неструктурированных треугольных сеток. Это необходимо для их более быстрой передачи по сетям и для непосредственного отображения на экране персонального компьютера пользователя.

Сжатие изоповерхности

Под триангулированной поверхностью будем понимать набор произвольно расположенных в трехмерном пространстве точек и набор определенных на них не пересекающихся треугольников, таких, что ни на одно ребро никакого треугольника не опирается более двух треугольников.

Далее рассматриваются два алгоритма сжатия изоповерхностей:

1. Сжатие зон однозначной проектируемости.
2. Сжатие последовательным исключением узлов.

Сжатие зон однозначной проектируемости

Рассмотрим алгоритм формирования триангулированной поверхности *G*, аппроксимирующей изоповерхность *F*, построенный таким образом, что бы практически полностью исключить необходимость хранения топологии поверхности *G*.

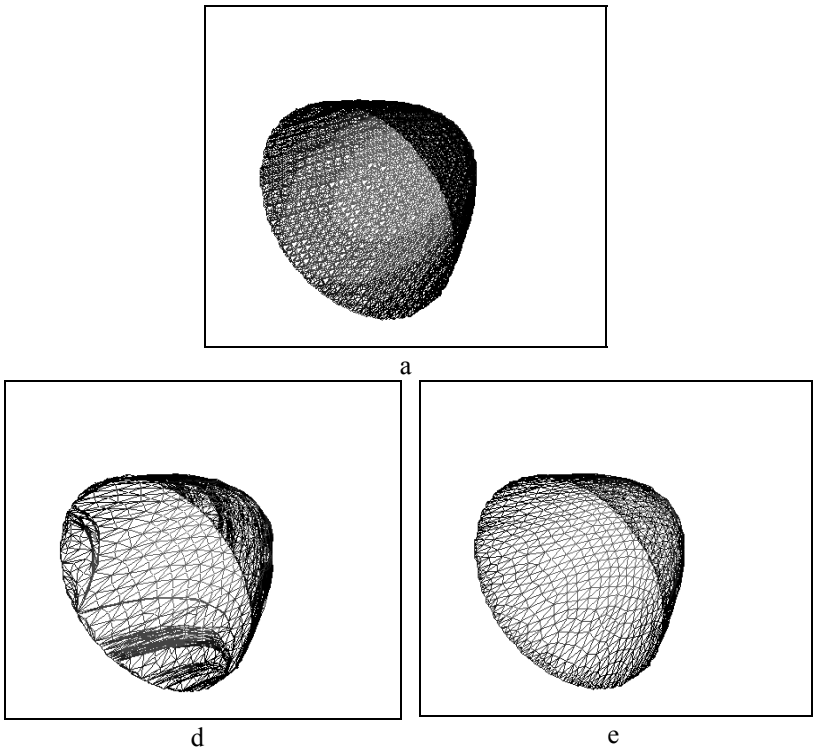


Рис. 2. Этапы сжатия изоповерхности.

Рассмотрим сжатие изоповерхности *F*, однозначно проецируемой на плоскость (*x,y*) (рис. 2a). В этом случае все точки, формирующие изоповерхность могут быть разделены на внутренние (соседи которых образуют простой цикл) и внешние. Удаляя все внутренние и часть внешних точек получим ряд опорных точек аппроксимирующей поверхности *G*. Теперь построим триангуляцию внутренней области,

добавляя внутрениие вершины триангуляции внутрь области последовательно по одной. При определении координат очередной точки будем использовать только информацию о координатах (*x,y,z*) опорных точек и координатах (*x,y,z*) уже добавленных точек, где *z* – координата пересечения нормали к плоскости (*x,y*) с изоповерхностью *F(x,y)*. Таким образом, для восстановления поверхности *G* необходимо передать на клиентскую часть только информацию (*x,y,z*) об опорных точках и значения (*z*) в добавленных точках. Нет необходимости передавать координаты (*x,y*) добавленных точек и топологию связей между ними, поскольку их можно полностью определить при восстановлении поверхности, просто повторив действия, выполненные при ее сжатии.

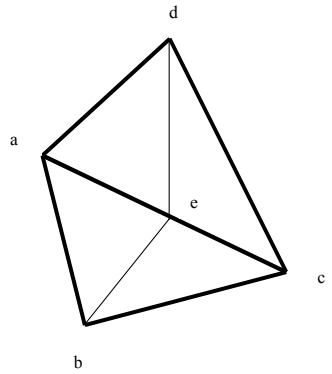


Рис.3. Этапы сжатия изоповерхности

Пример полученной таким методом поверхности приведен на рис. 2d. При построении поверхности (рис. 2d) использован простой алгоритм последовательного добавления точек:

1. выбор самого длинного ребра (*a,c*) в уже построенной части поверхности *G*;
2. добавление точки (*e*) в середину выбранного ребра (*a,c*);
3. соединение добавленной точки (*e*) ребрами с противолежащими вершинами треугольников (*b*),(*d*), опирающихся на ребро (*a,c*);
4. определение уровня *z* изоповерхности в точке (*e*).

Очевидно, что данный алгоритм позволяет воспроизвести изоповерхность на клиентской части, системы. Однако, простота

алгоритма приводит к тому, что в узкие длинные треугольники, построенные на опорных точках добавляется неоправданно много близких между собой точек. Использование более развитых алгоритмов позволяет получать вполне приемлемые результаты, ценою незначительного увеличения времени формирования поверхности *G*. Хорошие результаты дает добавление точки в центр тяжести треугольника максимальной площади с последующей корректировкой триангуляции в соответствии с критерием Делоне.

Изложенный подход позволяет получить выигрыш сразу в двух направлениях: во-первых, значительно уменьшается число точек (как правило достаточно нескольких сотен точек для передачи формы поверхности), во-вторых, дополнительно снижается объем передаваемых данных за счет отсутствия необходимости передачи информации о топологии поверхности *G*.

Для аппроксимации поверхностей, которые нельзя однозначно спроектировать ни на какую плоскость, разработан алгоритм разбиения произвольной триангулированной поверхности на связные фрагменты, каждый из которых может быть однозначно спроектирован на некоторую плоскость, не параллельную в общем случае никакой из координатных.

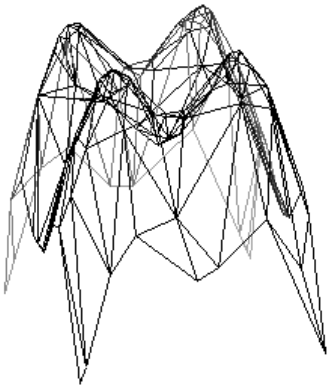
Представленный результат может быть значительно улучшен сокращением числа точек, определяющих границы изоповерхности и границы между зонами однозначности. Преимущество данного алгоритма заключается в простоте выполняемого анализа и в отсутствии необходимости хранения полной топологии аппроксимирующей поверхности. Необходимо хранить только параметры плоскостей проектирования, контура, очерчивающие зоны однозначной проектируемости, расстояния от плоскости проектирования каждой зоны до ее узлов и последовательность номеров треугольников в которые производилось добавление точек при формировании поверхности на стороне сервера, что позволяет использовать на стороне сервера произвольный алгоритм формирования поверхности.

Сжатие последовательным исключение узлов

Второй из разработанных алгоритмов не предполагает предварительного разбиения связной поверхности на части. Основная идея алгоритма заключается в удалении из исходной поверхности

некоторого количества узлов таким образом, что бы триангуляция, определенная на оставшихся точках, аппроксимировала исходную поверхность с требуемой точностью.

Точки исходных данных последовательно перебираются, при этом, для каждой точки проверяется, можно ли ее удалить, не нарушая требуемой точности аппроксимации. В случае удаления точки, соседние с ней вершины соединяются друг с другом так, чтобы образовавшийся многоугольник заполнился треугольниками. Удаленная точка заносится в список одного из этих треугольников,– ближайшего к ней. Соответствующие списки нужны для того, чтобы контролировать величину отклонения формируемой поверхности от исходной. Максимально допустимое отклонение заранее определено некоторой величиной **d**. Эта величина выбирается исходя из того, с какой точностью полученные данные должны аппроксимировать исходные, и принимается равной произведению L на e, где L – линейный размер отображаемых данных, а e – коэффициент желаемой точности (вполне допустимую точность дает его значение *0.01*). Для повышения качества аппроксимации удаление точек происходит итерационно, при этом на каждом шаге точки удаляются равномерно по всей поверхности, то есть временно не удаляются соседние с уже удаленными на текущем шаге. Таким образом, некоторые точки при просмотре пропускаются – возможность их удаления проверяется при последующих проходах по списку точек. По окончании очередного шага (если какие-либо точки были удалены) итерация повторяется. Таким образом, при каждом просмотре списка точек, некоторые из них удаляются вместе с треугольниками, опирающимися на эти точки, и новые треугольники добавляются.



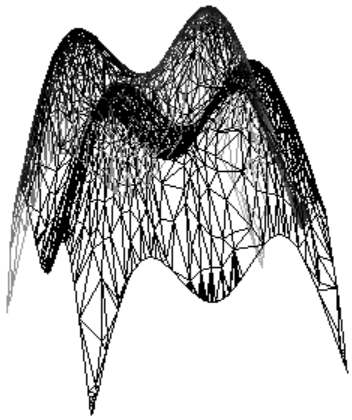


Рис. 4. Сжатие с точностью 2% Рис. 5. Сжатие с точностью 0,1%

К достоинствам алгоритма следует отнести хорошее качество аппроксимации с точки зрения внешнего восприятия построенной поверхности (рис. 5), универсальность и возможность сжатия с контролируемой точностью произвольных триангулированных несамопересекающихся, не обязательно связных, поверхностей. Изложенный алгоритм обладает свойством «локальности», – при обработке очередной точки анализируется информация только о небольшом фрагменте поверхности, непосредственно прилегающем к удаляемой точке, что позволяет говорить о высокой степени его внутреннего параллелизма и ожидать, что соответствующий алгоритм для многопроцессорных систем, построенный на принципах геометрического параллелизма, будет иметь высокую эффективность.

В заключении можно отметить высокую эффективность предложенных алгоритмов сжатия поверхностей, особенно с точки зрения их использования для обработки результатов вычислительных экспериментов, выполняемых с применением многопроцессорных систем, поскольку рассмотренные алгоритмы допускают эффективное распараллеливание и пригодны для обработки сеточных данных, объем которых сравним с объемом всей оперативной памяти используемой вычислительной системы.

Литература

1. Iakobovski M.V., Karasev D.E., Krinov P.S., Polyakov S.V. Visualisation of grand challenge data on distributed systems. In: Mathematical Models of Non-Linear Excitations, Transfer, Dynamics, and Control in Condensed Systems and Other Media. Proc. of a Symp. , June 27 – July 1 2000, Moscow, Russia (Ed. by L.A. Uvarova), Kluwer Academic // Plenum Publishers. – New York, Boston, Dordrecht, London, Moscow. ISBN 0-306-46664-3, 2001, pp. 71-78.
2. Iakobovski M.V., Krinov P.S. Visualisation of grand challenge data on distributed multiprocessor systems. // 5 International congress of mathematical modeling. Books of abstracts, V.1 // responsible for volume L.A.Uvarova – M.: «JANUS-K», 2002.
3. Кринов П.С., Поляков С.В., Якобовский М.В. Визуализация в распределённых вычислительных системах результатов трёхмерных расчётов. Труды четвёртой международной конференции по математическому моделированию. Под ред. Л.А.Уваровой. – М.: «Станкин», 2001г., т.2, с.126-133.

ПРИМЕНЕНИЕ ПАРАЛЛЕЛЬНЫХ СИСТЕМ ДЛЯ МОДЕЛИРОВАНИЯ В РЕАЛЬНОМ ВРЕМЕНИ

Р.К. Кудерметов, Н.Д. Пиза

Запорожский национальный технический университет, Украина
krk@zntu.edu.ua, ndp@zntu.edu.ua

Введение

На компьютерах, входящих в состав комплексных стендов (КС) и полунатурных моделирующих вычислительных комплексов (ПНМК), предназначенных для отработки систем управления космических аппаратов и других динамических объектов, а также на бортовых компьютерах (БЦВК), решение вычислительных задач выполняется в реальном или ускоренном масштабах времени. Это обусловлено необходимостью предоставления результатов решения задач для обмена с другими подсистемами через равные интервалы времени T, называемые базовыми тактами. За время базового такта необходимо выполнить моделирование движения объекта, процессы в приборах системы управления и внешней среде, что сводится к интегрированию

систем дифференциальных уравнений большой размерности. Как правило, для этой цели используются методы интегрирования Рунге-Кутта. С целью сокращения времени счета модели целесообразно рассмотреть возможность использования параллельных процессоров в КС, ПНМК и БЦВК.

Постановка задачи

Для интегрирования на параллельных процессорах разработаны параллельные методы интегрирования, в частности многошаговые и одношаговые блочные методы. Не теряя общности, достаточно рассмотреть только одношаговый двухточечный метод интегрирования, для которого рекуррентные формулы на сетке $\omega_h = \{t_n = n\,h,\ n = 1,\ 2, ...\}$ имеют вид [1,2]:

$$y_{n+r,0} = y_n + rhf_n,\ r = 1, 2,$$

$$y_{n+1,s+1} = y_n + \frac{h}{12}\left(5f_n + 8f_{n+1,s} - f_{n+2,s}\right), \qquad (1)$$

$$y_{n+2,s+1} = y_n + \frac{h}{3}\left(f_n + 4f_{n+1,s} + f_{n+2,s}\right),\ s = \overline{0, 2}\,.$$

Как видно из структуры приведенных формул, вычисления значений функций $f_{n+1,s}$, $f_{n+2,s}$ и получаемых решений $y_{n+1,s+1}$, $y_{n+2,s+1}$ можно выполнять параллельно. Необходимо оценить возможность использования параллельных блочных методов для интегрирования систем уравнений в реальном масштабе времени.

Основные результаты

Время, необходимое для выполнения одного блока интегрирования на каждом из параллельных процессоров по формулам (1), равно

$$T_{B2} = 7\,t_m\,n + 10\,t_s\,n + 8\,t_c + 4\,t_f\,, \qquad (2)$$

где n – размерность вектора неизвестных интегрируемой системы уравнений;

t_s, t_m – время, затрачиваемое на операции сложения и умножения, соответственно;

t_c – время, необходимое для пересылки вектора данных из n элементов между процессорами;

t_f – время вычисления правых частей уравнений.

Затраты времени на один шаг интегрирования методом Рунге-Кутта четвертого порядка на одном процессоре можно оценить как

$$T_{RK} = 8\,t_m\,n + 9\,t_s\,n + 4\,t_f\,. \qquad (3)$$

Сравним реализацию параллельного блочного метода интегрирования на двухпроцессорной вычислительной системе с методом Рунге-Кутта, реализованном на одном процессоре для выполнения моделирования в реальном масштабе времени.

Предположим, что время вычисления правых частей уравнений t_f значительно превышает суммарное время на остальные операции интегрирования, а время на пересылку t_c вектора длиной n между процессорами мало. Последнее предположение справедливо для мультипроцессорных систем (SMP). Тогда $T_{RK} \approx 4\,t_f$ и $T_{B2} \approx 4\,t_f$. Понятно, что шаг интегрирования h выбирается из соображений точности, а базовый такт T – на основании задачи управления.

Возможны два случая реализации интегрирования на базовом такте. Первый, когда базовый такт соизмерим со временем T_{RK} или T_{B2}, т.е. $T \approx 4\,t_f$, и интегрирование необходимо выполнять с шагом, равным длительности базового такта ($T = h$). Во втором случае базовый такт значительно превосходит время $4\,t_f$, тогда в базовом такте можно выполнить несколько шагов интегрирования методом Рунге-Кутта или блоков методом (1) и $T = 4m\,t_f$, где $m = 2,3,\ldots$

Рассмотрим в начале первый случай. За время базового такта можно выполнить только один шаг интегрирования методом Рунге-Кутта или один блок параллельным блочным двухточечным методом. Но метод Рунге-Кутта дает решение в точках $t_n + h$, а блочный – сразу в двух точках: $t_n + h$ и $t_n + 2h$ (рис.1). Поэтому использование блочного метода для рассматриваемого случая не имеет смысла, если правые части уравнений учитывают управляющие воздействия на каждом базовом такте.

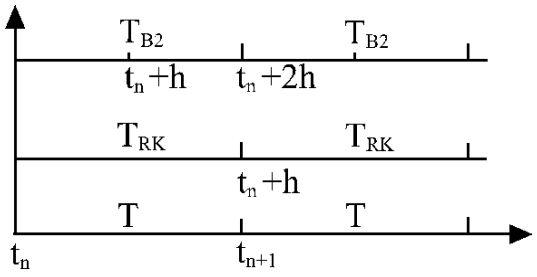


Рис. 1

С другой стороны, если допустимо интегрирование с шагом $h/2$, то применение блочного метода возможно. Известно [2], что порядок локальной ошибки одношаговых блочных методов равен $O(h^{k+2})$, где k – число точек в блоке, и, следовательно, совпадает с порядком локальной ошибки метода Рунге-Кутта четвертого порядка. Поэтому интегрирование с шагом $h/2$ двухточечным блочным методом при некоторых условиях, определяемых выбором шага интегрирования, может обеспечить большую или соизмеримую точность по сравнению с точностью метода Рунге-Кутта четвертого порядка при шаге h. Это видно из рис. 2, на котором представлены результаты исследования точности моделирования углового движения КА по методике, изложенной в [3].

Во втором случае, когда $T = 4m\,t_f$ и в течение базового такта возможно выполнение нескольких шагов интегрирования, например двух (рис.3), применение параллельных блочных методов может привести к существенному ускорению по сравнению с использованием метода Рунге-Кутта. Ускорение при этом в идеале равно двум и определяется по формуле

$$S = \frac{T_{RK}}{T_{B2}} \tag{4}$$

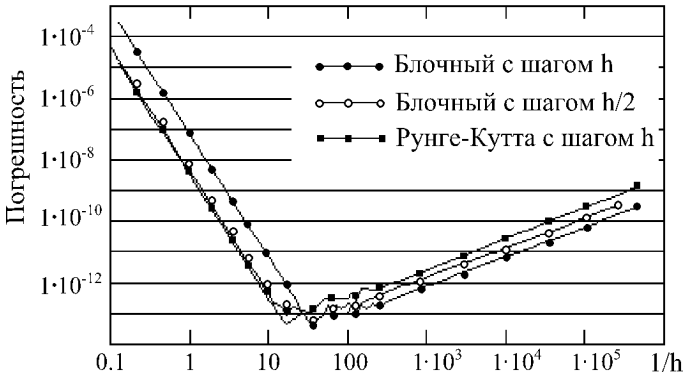


Рис. 2

Следует отметить, что, если обмен с внешними устройствами происходит только в начале или конце базового такта и в течение базового такта выполняется несколько блоков интегрирования, то возможно использование многошаговых параллельных блочных методов. Многошаговые блочные методы, как показано в [2], имеют порядок точности $O(h^{2k+1})$.

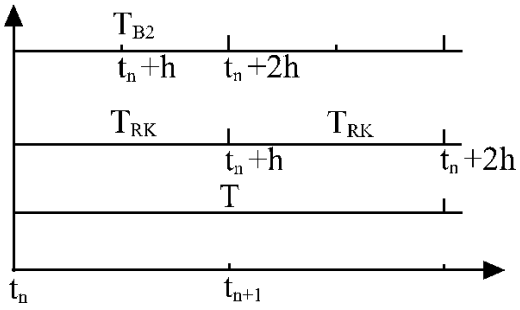


Рис. 3

Выводы

Таким образом, применение параллельных процессоров и параллельных блочных методов интегрирования для моделирования динамических систем в реальном масштабе времени может дать значительное ускорение вычислений и, следовательно, высвобождение

процессорного времени для решения других задач в составе ПНМК, КС и БЦВК.

Литература

1. Системы параллельной обработки / Под ред. Д. Ивенса. – М.: Мир, 1985. – 416 с.
2. Фельдман Л.П., Дмитриева О.А. Разработка и обоснование параллельных блочных методов решения обыкновенных дифференциальных уравнений на SIMD-структурах // Наукові праці Донецького національного технічного університету. Вип. 29. – Донецьк, 2001. С. 70-79.
3. Пиза Н.Д., Кудерметов Р.К. Применение параллельных блочных методов для моделирования движения космического аппарата // Вісник технологічного університету Поділля. – 2003. – №3, Т1. – С. 93-96.

СОЗДАНИЕ КЛАСТЕРНЫХ РЕСУРСОВ ДЛЯ ХИМИЧЕСКИХ ИССЛЕДОВАНИЙ. ТЕСТИРОВАНИЕ АППАРАТНОЙ ПЛАТФОРМЫ НОВОГО ПОКОЛЕНИЯ ДЛЯ УЗЛОВ

М.Б. Кузьминский, М.Н. Михайлов

Институт органической химии РАН, г. Москва

При проведении химических исследований в требующих высокопроизводительных вычислительных ресурсов областях, таких, как квантовая химия, молекулярная динамика и молекулярная механика, применение кластеров оказывается достаточно эффективным. Эти задачи относятся к приложениям с плавающей запятой, соответственно при выборе узлов по критериям производительности и отношения стоимость/производительность можно ориентироваться прежде всего на известные тесты, например, Linpack или SPECfp2000. Так, в [1] была найдена корреляция между SPECfp_base2000 и производительностью ряда микропроцессоров (МП) на тестах Gaussian-98. Однако известно, что, например, в некоторых реальных приложениях квантовой химии (т.н. «in-core» методы) применение оперативной памяти (ОП) очень большой емкости с хранением основных массивов в ОП способно существенно поднять производительность. Естественно, в этом случае необходимо

применение 64-разрядных МП, а сопоставление производительности на основе SPECfp2000 становится некорректным. Кроме того, 64-разрядная адресация позволяет рассчитывать системы с большими размерностями.

Недавно появились новые 64-разрядные МП AMD Opteron с существенно более высоким отношением производительность/стоимость, чем у альтернативных 64-разрядных платформ. При построении кластеров МП Opteron по уровню производительности и отношению стоимость/производительность являются конкурентами как 64-разрядных МП, так и 32-разрядных Intel Xeon. Известно, что на тестах SPECcpu2000 Opteron/1.8 ГГц немного опережает Xeon/3.06 ГГц, хотя немного уступает Pentium 4/3 ГГц с FSB 800 МГц.

В [1] найдено, что типичные приложения квантовой химии на тестах Gaussian-98 лимитируются обращениями в ОП. Микроархитектура Opteron способствует пониженным задержкам при обращении в ОП и повышенной пропускной способности (ПС) ОП в SMP-конфигурациях за счет независимых каналов доступа МП в ОП, в отличие от общей шины в Intel Xeon. Поэтому оценки производительности Opteron весьма актуальны, и в Центре компьютерного обеспечения химических исследований РАН (ЦКОХИ) были проведены измерения производительности 2-процессорной SMP-системы на базе Opteron как на тестах универсального характера (Linpack, n=100 и 1000; STREAM и lmbench (2.0.4)), так и на конкретных прикладных задачах квантовой химии. Для указанных универсальных тестов необходимо применение таймеров с высоким разрешением, подобно использованной нами ранее [2,3] подпрограммы-таймера на базе RDTSC. Применение команды RDTSC в Opteron имеет своими особенностями малое время задержки и возможность внеочередного выполнения, что при обычном использовании может иногда привести к неверным результатам. В связи с этим в соответствующих случаях мы воспользовались системным вызовом gettimeofday ОС Linux, который корректно работает с RDTSC и в наших тестах на Opteron не привносит существенной «собственной задержки».

В тестах использован Opteron/1.6 ГГц с материнской платой RioWorks HDAMA с двухканальной ОП DDR266 CL2.5. Тесты проведены также на Athlon MP1800+ (с близкой тактовой частотой

1533 МГц, с одноканальной DDR266), Pentium 4/1.8 ГГц (кэш L2 256K, RDRAM PC800) и Pentium III 1266 МГц (кэш L2 512K, с двухканальной ОП PC133); проведено сопоставление с другими литературными данными (Табл.1–3). На тестах Linpack сопоставлены компиляторы pgf90 5.0 (бета-версия), g77-3.3 и Intel ifc 7.1, а также библиотеки AMD acml-0.9 (бета-версия), Intel MKL-6.0 и Atlas 3.5.1. Сервер с Opteron работал под ОС UnitedLinux/SuSE Linux Enterprise Server for Opteron (бета-версия).

Таблица 1.

Сопоставление процессоров узлов кластера на универсальных тестах

Процессоры	Linpack (MFLOPS)		Пиковое значение, GFLOPS	Задержка обращения в память, нс
	n=100	n=1000		
Xeon/3.06 (1)	1190	2355	6.1	102 (2)
Pentium 4/1.8	761	1831	3.6	-
Athlon MP1800+	732	1623	3.1	191 (2)
Opteron/1.6	818	2103	3.2	109
Itanium 2/1.0	1102	3534	4.0	156 [4]

Примечания.

(1) с DDR266/CL2.0. Данные Linpack приведены для Pentium 4/2.53 ГГц. Данные для Athlon, Opteron и Pentium 4/1.8 ГГц получены авторами, остальные взяты из официальной таблицы тестов на сайте //netlib2.cs.utk.edu (наши данные для Athlon опубликованы ранее [2,3] и помещены в официальную таблицу. Для Pentium 4 использован ifc с ключами –O3 –tpp7 –xW –ip и библиотека MKL, для Opteron – g77 с ключами –O3 –m32 –mfpmath=sse –malign –march=athlon-xp –funroll-loops –fomit-frame-pointer и библиотека Atlas. Для n=1000 при использовании на Opteron старых кодов, оптимизированных под Athlon, ускорение по сравнению с AthlonMP составило 1.3-1.4 (зависит от библиотеки).

(2) Данные //www.tecchannel.de/hardware/1164/15.html: для Xeon – с DDR266/CL2.0; для Athlon – приведены для Athlon MP2600+, с DDR266/CL2.0 .

Таблица 2.

Эффективность применения некоторых 64- и 32-разрядных микропроцессоров (в SMP-конфигурациях) на задачах квантовой химии с небольшим объемом ввода-вывода

Тесты Gaussian-98	N CPU	Процессорное время расчета, сек				
		AMD		Intel		SGI Origin 3000
		Opteron 1.6 ГГц	Athlon MP1800+ 1533 МГц	Pentium III 1266 МГц	Itanium 733 МГц	R14000A 600МГц
test178(RHF) (симметрия D3H)	1 2	72 43	131 92	117 84	80 -	94 -
test178 (без симметрии)	1 2	358 185	580 342	535 341	- -	- -
C3H11O4/ MP2	1	170	311	348	189	247
CH6N2/MP4	1	1977	3030	3759	2070	3158
test397(DFT)	1 2	21763 11077	36027 19702	- -	27160 -	29298 -

Примечания. Для Opteron, Athlon и Pentium III нами использована двоичная версия Gaussian-98 Rev.A11 [5], транслированная с помощью pgf77-4.2 с оптимизацией под Pentium III. Данные для Itanium и R14000 взяты из [6]. Значения SPECfp_base2000 для Opteron, Athlon MP, Itanium и R14000A равны соответственно 1029, 587, 623 и 499 (см. //www.specbench.org).

По данным SPECfp2000 и наших тестов прикладных программ, производительность Opteron с плавающей запятой существенно выше, чем в Athlon XP. Но пиковая производительность Opteron меньше, чем у Pentium 4, из-за гораздо более низкой тактовой частоты, и на тестах, имеющих высокую долю операций с плавающей запятой (например, тестах Linpack), Opteron может сильно уступать Xeon/Pentium 4 (Табл.1).

Измеренная нами задержка доступа в ОП на тестах lmbench составила 109 нс (табл.1). Это лучше, чем в Athlon MP, но всего лишь близко к результатам Xeon/3.06 ГГц. Для Itanium 2 с набором микросхем HP zx1 задержка равна 156 нс.

Таблица 3.

Эффективность применения 64- и 32-разрядных x86-совместимых микропроцессоров на задачах квантовой химии с большим объемом ввода-вывода

Тесты Gamess-US (conventional) [7]	n CPU	Процессорное время расчета, сек		
		Opteron 1.6 ГГц	Athlon MP1800+ 1533 МГц	Pentium 4 2.8 ГГц
test178(RHF)	1 2	49/57 30/47	96/102 -	68/71[(1)] 41/53[(2)]
test397(DFT)	1	11446/28049	-	13996
C3H11O4/MP2	1 2	83,6/84,3 69/108	132/102 -	93[(3)] -

Примечания. Через слэш приведено старт-стопное время, которое в conventional-методах из-за большого ввода-вывода гораздо больше (лимитирует время ожидания ввода-вывода).

(1) Все данные для Athlon получены при трансляции GAMESS- US с помощью ifc 7.1 с ключами -O3 -tpp6 -ip, а для остальных процессоров – с ключами -O3 -tpp7 -xW -ip. При трансляции с ключами -O3 -tpp6 с последующим прогоном на Pentium 4 время выполнения равно 71/75 сек.

(2) Для Pentium 4 приведены данные не для SMP-сервера, а для двух однопроцессорных узлов кластера, соединенных Fast Ethernet со связыванием двух каналов.

(3) При трансляции с ключами -O3 -tpp6 -ip время выполнения составляет 104/113 сек.

На тестах ПС ОП (STREAM) удалось достигнуть ПС 2.3-2.5 Гбайт/с, что более чем в 2 раза больше, чем у Athlon MP1800+ с одноканальной DDR266. Конечно, Pentium 4/3 ГГц с FSB 800 МГц и двухканальной DDR400 будут далеко впереди. Однако известно, что у Pentium 4 в двухпроцессорных системах из-за конкуренции на системной шине параллельное выполнение двух нитей тестов STREAM немного снижает суммарную ПС, в то время как на Opteron нами обнаружен рост суммарной ПС примерно в 1.8 раза.

Известные приложения квантовой химии – 32-разрядные программы Gaussian-98 и Gamess-US, отличающиеся высокими требованиями к производительности с плавающей запятой и различным поведением с точки зрения локализации в кэше,

требованиями к ПС ОП и т. д. в зависимости от метода расчета, на Opteron/1.6 ГГц по процессорному времени ускорились в 1.5-1.9 раза по сравнению с Athlon MP1800+ (на одном процессоре). При этом ускорение при распараллеливании программ на 2 процессора SMP-сервера для Opteron оказалось заметно выше, чем для Athlon MP (табл.2,3). Однако ускорение Gamess-US в Opteron по сравнению с Pentium 4/1.8 ГГц оказалось всего около 1.2-1.4. Это может быть связано с худшей по сравнению с Gaussian-98 оптимизированностью исходных текстов. Интересно, что ускорение Gamess-US при оптимизации под Pentium 4 по сравнению с оптимизацией под Pentium III составило 4-12% при выполнении на Pentium 4/1.8 Гц (Табл.3). Неожиданно высокие результаты для Pentium III Tualatin/1266 МГц (Табл.2) связаны, вероятно, с более высокой емкостью кэш-памяти второго уровня в этом МП (512 Кбайт, в 2 раза больше, чем в Athlon MP).

Приведенные результаты показывают, что хотя определенная корреляция SPECfp_base2000 и производительности МП на задачах квантовой химии наблюдается, имеется целый ряд исключений, зависящих от специфики задач и микроархитектуры МП (что и следовало ожидать, учитывая разнообразие расчетных методов). Соотношение производительностей Opteron и Pentium 4/Xeon достаточно сильно зависит от теста.

Выбор каналов для соединения узлов кластера в определенной степени не зависит от аппаратной платформы узлов и был рассмотрен нами ранее [3]. Однако для эффективного применения кластерных ресурсов (особенно в условиях сильно различного поведения приложений, таких как Gaussian-98 и Gamess-US, при распараллеливании) необходима эффективная организация вычислительного процесса.

С этой целью в ЦКОХИ проведена разработка новой версии пользовательских интерфейсов с прикладными программами и системой пакетных очередей Sun Grid Engine. При этом обеспечена высокая переносимость на новые прикладные программы (текущая реализация включает Gaussian-98 и Gamess-US) и пакетные системы; аутентификация пользователей с контролем доступа к каждой прикладной программе; автоматический перенос контрольных точек между узлами; возможности восстановления заданий при сбоях

системы пакетных очередей; автоматическое сжатие файлов при передаче данных заданий между узлами. Используется простой язык управления заданиями, содержащий заказы определенных ресурсов. Данные интерфейсы используют средства электронной почты и ftp; Web-интерфейс может надстраиваться над ними. Реализована также система команд оператора в стиле IBM JES2, общая с также используемой в ЦКОХИ средой Generic NQS.

Литература

1. Kenny Yu. J.-S., Chin-Hui Yu., J.Chem . Inf. Comput. Sci., 2003, v. 42, p. 673.
2. Кузьминский М.Б. Открытые системы, 2002, N9, с. 10.
3. М.Б.Кузьминский, Мендкович А.С. и др. Высокопроизводительные параллельные вычисления на кластерных системах. Сб. материалов II Международного научно-практического семинара, 26-29 ноября 2002 г., Изд-во ННГУ, Н. Новгород, 2002, с.169.
4. Кузьминский М.Б. Открытые системы, 2003, N 3, с. 10.
5. Gaussian 98, Revision A.11.3, M.J. Frisch, G.W. Trucks, H.B. Schlegel, G.E. Scuseria, e.a., Gaussian, Inc., Pittsburgh PA, 1998.
6. SGI Computational Chemistry Applications Performance Report. Spring 2002, Silicon Graphics, Inc., 2002.
7. Schmidt M.W., Baldridge K.K., Boatz J.A., Elbert S.T. e.a., J. Comput. Chem., 1993, v. 14, p. 1347.

УПРАВЛЕНИЕ ВЫЧИСЛИТЕЛЬНЫМИ РЕСУРСАМИ ВЫСОКОПРОИЗВОДИТЕЛЬНОГО ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА МВС-1000/М СИБИРСКОГО СУПЕРКОМПЬЮТЕРНОГО ЦЕНТРА

Н.В. Кучин

ИВМиМГ СО РАН, г. Новосибирск
kuchin@ssd.sscc.ru

Как только появляется вычислительный ресурс, которой нужно делить между многими пользователями или между многими задачами одного пользователя встает задача организации доступа к ресурсу и эффективного его использования. Таким уникальным ресурсом являются сервера RM600 E30 и высокопроизводительный

вычислительный кластер МВС-1000/М Сибирского Суперкомпьютерного Центра (ССКЦ) [1]. Пользователи ССКЦ имеют возможность, отладив свою параллельную программу на нашем МВС-1000/М, для проведения крупномасштабных вычислительных экспериментов перенести ее на аналогичный суперкомпьютер московского Межведомственного СуперКомпьютерного Центра с 768 процессорами [2]. Кластер, включая МВС-1000/М, это сеть компьютеров. Создание центров коллективного пользования сделало актуальным пакетную обработку заданий на сетях компьютеров [3], а также создание и эксплуатацию систем управления вычислительными ресурсами [4]. В статье автор использовал материалы своего отчета по гранту РФФИ 01-01-01051-а «Сравнительный анализ систем пакетной обработки».

1. Системы управления вычислительными ресурсами

Базовые понятия управления вычислительными ресурсами рассмотрены в [4] на примере CODINE. Эта система интересна тем, что она создана как коммерческая версия с расширенными возможностями на основе свободно распространяемой системы пакетной обработки DQS (Distributed Queueing System) [5], которая установлена и работает на комплексе RM600 E30 ССКЦ. В настоящее время CODINE превратилась в Sun Grid Engine[6] и наряду с PBS (Portable Batch System) [7].входит в число наиболее развитых и популярных систем пакетной обработки.

Система управления вычислительными ресурсами (Resource Management Systems – RMS) позволяет администратору планировать и управлять использованием вычислительными ресурсами. Типичная система управления заданиями включает следующие возможности:

- **поддержку пакетной обработки**;
- **поддержку параллельных вычислений**: системы параллельного программирования, такие как Parallel Virtual Machine (PVM) и Message Passing Interface (MPI с его реализациями mpich и LAM) позволяют разрабатывать и выполнять параллельные программы на языках C, C++, f77 и f90 как на параллельных ЭВМ, так и на сети рабочих станций. Таким образом кластер может работать как параллельный компьютер.

- **контрольные точки и миграция процессов**: контрольные точки это обычный метод сохранения текущего состояния задачи. В случае аварийной ситуации на компьютере будут потеряны только результаты вычислений, полученные с момента создания последней контрольной точки. Миграция процессов это возможность перезапуска процесса на другой машине кластера без рестарта всей задачи. Обычно миграция процессов применяется для балансировки вычислительной нагрузки на узлах кластера.
- **балансировка вычислительной нагрузки** предполагает распределение вычислительной нагрузки между узлами кластера таким образом, чтобы каждый узел выполнял эквивалентный объем работы.
- **лимиты времени выполнения**: задания планируются исходя из требуемых им ресурсов, таких как процессорное время, объём оперативной памяти и т.д. При этом короткие задания обычно имеют приоритет перед длинными. Лимиты времени выполнения (Run-Time Limits) устанавливаются для того, чтобы заказанные пользователем ресурсы соответствовали фактическому использованию. В частности превышение лимита может означать зацикливание задачи. Как правило при превышении лимита задача завершается принудительно.

Компоненты и архитектура системы управления ресурсами (Resource Management Systems)

Система управления ресурсами в целом предназначена для управления ресурсами как одиночного компьютера, так и сети компьютеров. Базовые требования это подключение компьютеров в сеть и поддержка многопользовательского и многозадачного режимов работы. В большинстве случаев в качестве стандартной операционной системы используют UNIX.

- *Пользовательский интерфейс*

Любая популярная система управления ресурсами как минимум позволяет вводить команды из командной строки. Типичные пользовательские команды это:

- передача задания в систему управления ресурсами для дальнейшего выполнения,
- просмотр состояния заданий (в очереди на выполнение, выполняются, завершились),

- уничтожение запущенных заданий.

- *Административная среда*

  Задача администратора в системе управления ресурсами это:
  - описание машинных характеристик для вычислительных узлов (хостов),
  - определение классов заданий для исполнения и правил выбора наиболее подходящей машины для исполнения выбранного задания,
  - определение для пользователя разрешения на доступ к использованию ресурсов,
  - определение предельного размера ресурсов (процессорного времени, объема оперативной памяти и т.д.) для заданий и пользователей,
  - определение оптимальной стратегии для распределения заданий по вычислительным узлам, например с учетом их текущей загрузки,
  - управление и обеспечение корректного функционирования системы управления ресурсами.

Некоторые системы управления ресурсами обладают специальной графической средой для администрирования RMS; но по крайней мере команды управления системой управления вычислительными ресурсами могут выдаваться из командной строки ОС UNIX.

Ресурсы, задания, стратегии

- *Задания*

С точки зрения RMS задание это набор программ: это может быть и одна программ, и несколько взаимодействующих программ, также задание может включать команды операционной системы. Существует несколько типов заданий:
- пакетные задания, которые не требуют непосредственного взаимодействия с пользователем в процессе своего решения;
- интерактивные задания, которые требуют диалоговый ввод данных во время выполнения;
- параллельные задания это задания, чьи подзадачи (параллельные ветви) выполняются одновременно на нескольких машинах кластера.
- задания с контрольными точками периодически сохраняют своё состояние в файл (файлы) и тем самым могут быть

прерваны оператором или самим пользователем в любое время. После перезапуска такие задачи продолжают своё выполнение с состояния, записанного в последнюю по времени контрольную точку. С некоторыми ограничениями такие задания могут в процессе своего выполнения мигрировать между машинами кластера.

- *Вычислительные ресурсы*

К ресурсам относятся число и тип процессоров (вычислительных модулей), требуемая/доступная оперативная память, дисковая память, процессорное время, периферийные устройства. При конфигурировании системы пакетной обработки описываются доступные ресурсы. Задания включают в себя список требуемых для выполнения задания ресурсов. В процессе выполнения задания ресурсы используются и соответственно их доступный объём уменьшается. Система управления вычислительными ресурсами следит за тем, чтобы объем используемых выполняемыми заданиями ресурсов не превышал объём ресурсов, выделенных на кластере и конкретных машинах кластера.

- *Стратегии*

Кроме разбиения заданий на классы и соответствующим конфигурированием очередей система управления вычислительными ресурсами может предоставлять более развитое средство управления использованием вычислительных мощностей. Часто такого типа средства называют *policies*. Этот термин мы переводим как *стратегия* (схема) выделения / распределения вычислительных ресурсов.

Сюда входят стратегии использования ресурсов и планирования заданий.

- Стратегии планирования заданий

Динамические стратегии распределения (использования) вычислительных ресурсов оказывают влияние как на использование (потребление) вычислительных ресурсов заданиями как на этапе их выполнения, так и на этапе распределения заданий по вычислительным узлам: где, когда и какое задание запускать на выполнение.

Современные RMS могут поддерживать различные стратегии планирования заданий, такие как First-Come-First-Served (планирование заданий на выполнение в порядке их поступления в очередь), Select-Least-Loaded (выбор наименее загруженного), Select-

Fixed-Sequence (выбор фиксированной последовательности заданий), а также комбинацию этих стратегий.

- Стратегии использования ресурсов

Можно выделить распределение ресурсов (квотирование), функциональный подход – выделение ресурсов с учетом важности функций подразделения. И наконец выделение ресурсов с учетом предельного времени завершения заданий (deadline). Остановимся подробнее на квотировании, поскольку именно эта схема распределения ресурсов наиболее подходит для ССКЦ. Вычислительные ресурсы (машинное время, дисковое пространство и т.д.) делятся между структурными подразделениями (институтами, отделами), проектами и пользователями. Существенно то, что при таком подходе к распределению вычислительных ресурсов задается интервал планирования (например, неделя или месяц) и в течение этого интервала RMS поддерживает данную схему (и данные недельные, месячные квоты) распределения ресурсов. Поэтому учитывается фактическое использование машинного времени на протяжении всего интервала и RMS компенсирует (пользователям, подразделениям) недоиспользованное машинное время и тормозит задания тех, кто перебирает свои квоты. Такой подход обычно используется в случае, когда несколько различных организаций вложили свои средства в создание и эксплуатацию кластера, или поддерживаемые на кластере проекты требуют определенных объемов вычислительных ресурсов, либо для того чтобы разделить вычислительные мощности более-менее справедливым образом между всеми (организациями, подразделениями) участниками.

Вмешательство администратора *(manual override)* не является автоматической стратегией, но может поддерживаться современными RMS. Администратор может добавить ресурсы определенному заданию, пользователю, отделу, проекту или классу заданий.

2. Система управления заданиями для МВС-1000/М

Высокопроизводительный вычислительный кластер МВС-1000/М состоит из вычислительных модулей (ВМ) и управляющего модуля. Каждый вычислительный модуль CS20 представляет собой двухпроцессорную SMP – систему на базе микропроцессоров DEC Alpha (Alpha 21264A (EV67/833 МГц, 4 Мбайта кэш L2)) с 2 Гбайтами

оперативной памяти. Модули объединяются в многопроцессорную вычислительную систему (МВС) с помощью специализированного коммуникационного оборудования фирмы Myrinet, используемого для обмена сообщениями между выполняемыми параллельными процессами и обычного Fast Ethernet, который используется для управления узлами кластера и поддержки сетевой файловой системы.

Модули работают под управлением ОС LINUX (Red Hat 6.2); система управления прохождением задач для МВС-1000/М разработана в ИПМ им. Келдыша [8,9].

На МВС-1000/М установлены трансляторы Fortran и C/C++ фирмы Compaq, библиотека параллельного программирования mpich-gm.

Основные принципы, заложенные в логику работы системы управления заданиями:

1. Предусмотрена работа только с параллельными задачами, использующими библиотеку параллельного программирования mpich-gm для myrinet. Последовательное задание запускается как параллельное, но запрашивающее только один процессор.
2. На время счета вычислительные узлы выделяются параллельной программе в монопольное использование, т.е. на одном вычислительном модуле могут работать два параллельных процесса одной параллельной программы, но не больше. Мультипрограммирование на пользовательском уровне не предусмотрено, поэтому невозможно запустить параллельных процессов больше, чем реальных процессоров в кластере.
3. Управляющий модуль для вычислений не используется. На нем размещаются пользовательские каталоги / директории и часть общесистемного программного обеспечения кластеры (включая очередь заданий), которые по протоколу NFS экспортируются на вычислительные узлы.
4. По времени счета задачи делятся на отладочные (короткие), пакетные (длинные) и фоновые (очень длинные). Для фоновых задач пользователь указывает минимальный квант времени, в течение которого задача не может быть прервана. Этот квант не может быть больше, чем максимальное время для пакетного задания. Предполагается, что за этот квант времени задача выполнит осмысленный кусок работы и сохранит требуемую информацию в промежуточных массивах, создаст контрольную

точку. Если по истечении кванта других заданий в очереди не будет, то считающаяся фоновая задача получит следующий квант и продолжит свое выполнение не прерываясь. В противном случае система управления заданиями прервет выполнение задания и вновь поставит его в очередь; таким образом фоновое задание будет считаться порциями (квантами) до тех пор, пока не отсчитает все заказанное время.

Приоритет заданий в очереди определяется системой управления заданиями с учетом заказанного времени счета и уже использованным заданиями пользователя временем.

Далее цитата из [8]:

«Планирование очередей в каждый момент времени производится в соответствии с параметрами текущего *режима планирования*. Режим планирования определяется следующими параметрами:

- дата и время включения режима;
- максимальное время, отведенное для отладочных задач;
- максимальное время, отведенное для пакетных задач;
- число процессоров, которые «резервируются» для отладочных задач;
- шкала приоритетов пользователей».

Разработанная в ИПМ им. Келдыша система управления заданиями достаточно узко специализирована, полностью ориентирована на кластер МВС-1000М и не переносима на параллельные компьютеры другой архитектуры, такие как, например RM600 E30. Алгоритмы планирования заданий построены на вычисляемых приоритетах. Система поддерживает ряд ограничений, не позволяющих одному или нескольким пользователям монополизировать вычислительные мощности (ограничение на число заданий, которые один пользователь может одновременно иметь в счете и/ или на стадии выполнения).

При планировании учитывается только один вычислительный ресурс – процессорное время. Это нормально для однородных систем; но через какое- то время системы будут модернизироваться, наращиваться и вычислительные модули могут отличаться быстродействием процессоров, объемами оперативной и дисковой памяти, скоростями обменов между модулями. Поэтому при

выделении вычислительных модулей заданиям придется как-то учитывать и эти характеристики.

Система управления заданиями, кроме управления пакетными заданиями, поддерживает целостность МВС и функционирование комплекса как единой многопроцессорной вычислительной системы (Single System Image).

В текущей версии распределение машинного времени (квотирование) между организациями и/или пользователями пока не поддерживается. Но система развивается, в ней появляются новые возможности. Так в версии 2.01 появились возможности по управлению локальной дисковой памятью на вычислительных модулях.

Заключение

Любая высокопроизводительная вычислительная техника для нормальной эксплуатации требует специального помещения с кондиционерами и источниками бесперебойного питания, обеспечение сетевого доступа к вычислительным ресурсам и наличие квалифицированных специалистов для эксплуатации и сопровождения технических средств и программного обеспечения, а также модернизации (включая программное обеспечение) и консультационного обслуживания пользователей. Дело это дорогое и единственно разумное решение это централизация имеющихся ресурсов и обеспечение коллективного доступа к вычислительным мощностям.

Литература и ссылки

1. http://www2.sscc.ru/ Web-site Сибирского СуперКомпьютерного Центра.
2. http://www.jscc.ru/ Web-site Межведомственного СуперКомпьютерного Центра.
3. http://www.osp.ru/os/2000/07-08/010.htm Журнал «Открытые Системы», #07-08/2000 Виктор Коваленко, Евгения Коваленко Пакетная обработка заданий в компьютерных сетях.
4. Ferstl F., Job and Resource Management Systems, High Performance Cluster Computing: Architectures and Systems, vol. 1, pages 499-533, Prentice Hall – PTR, NJ, USA, 1999.

5. http://www.scri.fsu.edu/~pasko/dqs.html DQS – Distributed Queueing System.
6. http://www.sun.com/gridware/ Sun Grid Engine.
7. http://www.OpenPBS.org Portable Batch System.
8. Руководство системного программиста (администратора) системы управления прохождением задач МВС-1000/М.
9. Лацис А. Как построить и использовать суперкомпьютер. *«Бестселлер»*, 2003. http://www.sharebook.ru/super/

СИСТЕМА УДАЛЕННОГО ДОСТУПА К ВЫЧИСЛИТЕЛЬНОМУ КЛАСТЕРУ

Д.Ю. Лабутин

Нижегородский государственный университет им. Н.И.Лобачевского
dmitry@incub.ru

С каждым годом увеличивается мощность компьютеров, но очевидно, что классическая фон-неймановская организация вычислений уже не обеспечивает требуемой производительности в некоторых задачах. Это обусловлено как физическими ограничениями (скорость света), так и экономическими. Поэтому стимулируется развитие параллельных вычислений. Тот факт, что параллельные вычисления до сих пор не получили должного распространения, объясняется техническими трудностями (необходимость высокоскоростных каналов связи между вычислительными комплексами или процессорами), алгоритмическими трудностями (необходимость разработки алгоритмов, учитывающих параллелизм), и, в особенности, отсутствием неких стандартных подходов и программных средств для организации параллельных вычислений.

С другой стороны, просто купить и установить вычислительный кластер вовсе не достаточно. Можно даже настроить программное обеспечение, необходимое для проведения вычислительных экспериментов. Но и при этом эффективно использовать кластер не получится. Вот, основные проблемы, которые при этом возникают:

- возможные конфликты в процессе запроса вычислительных мощностей во время проведения экспериментов (может случиться так, что во время запуска приложения на определенных

компьютерах, на них уже исполняются задачи других пользователей);

- необходимость личного присутствия разработчика на вычислительной площадке во время запуска экспериментов.

Для решения проблемы «конфликтов» разрабатывается центральная система управления вычислительным кластером. Благодаря этому все запросы на исполнения программ будут обрабатываться системой управления, а вычислительные мощности распределяться согласно запросам разработчиков программ и других условий.

Для решения проблемы личного присутствия разработчика на вычислительной площадке было решено использовать уже практически ставший стандартным подход – управление через Internet. Разрабатывается Web-интерфейс, который взаимодействует с описанной выше центральной системой управления вычислительным кластером. Благодаря популярному подходу к дистанционному управлению различными устройствами подключенными к всемирной компьютерной сети с помощью обычного Web-броузера, разработчик сможет производить запуск экспериментов с любого компьютера, подключенного к Internet'у, независимо от установленной на нем операционной системы.

Web-интерфейс разрабатывается как отдельный независимый модуль системы управления кластером. Взаимодействие с центральной системой производится через механизм сокетов (socket). Для этого был разработан специальный протокол обмена данными. Благодаря такому подходу можно будет разработать и stand-alone приложение, с помощью которого можно выполнять те же действия, что и через Web-интерфейс.

Основные функциональные возможности менеджера доступа

Аутентификация пользователей при работе с Web-интерфейсом управления вычислительным кластером. Каждый пользовать в начале работы должен ввести «Имя пользователя» и «пароль», известные только ему. После этого можно однозначно сопоставлять производимые действия на кластере с конкретным пользователем.

Выделение независимого файлового пространства. Каждому пользователю выделяется индивидуальное место для хранения файлов, необходимых для проведения вычислительных экспериментов. При

этом накладываются определенные ограничения на хранимые файлы (максимальный суммарный размер хранимых файлов, максимальный размер каждого файла в отдельности), индивидуальные для каждого пользователя системы.

Система задач. После того как пользователь загрузил необходимые файлы для проведения вычислительных экспериментов, он создает именованные задачи, т.е. задает необходимые параметры (имя задачи, исполняемый модуль, приоритет, имя файла с результатами, необходимое количество вычислительных мощностей, параметры командной строки для исполняемого модуля). Затем пользователь может ставить задачу в очередь на исполнения неоднократно, не утруждая себя повторным заданием параметров. В случае необходимости владелец задачи может снять ее с исполнения или убрать из очереди.

Мониторинг. После того, как одна или более задач поставлены в очередь на исполнение, пользователь может посмотреть их состояние (выполняется, ожидает выполнения в очереди, выполнилась). Так доступна информация о загруженности вычислительного кластера в целом.

Протоколирование действий. Все производимые пользователем системы действия протоколируются. Анализируя протокол можно определить, какие системные ресурсы выделяются тому или иному пользователю.

Административный интерфейс. Также имеется интерфейс работы с системой, предназначенный только для администраторов, где они могут производить такие действия, как добавление и удаление пользователей системы, изменение параметров, задающих ограничения на хранимые файлы, контроль хранимых пользователями файлов и т.п.

КРАТКИЙ ОБЗОР МЕТОДОВ МОНИТОРИНГА СОСТОЯНИЯ КЛАСТЕРОВ

И.В. Лопатин

Нижегородский государственный университет им. Н.Лобачевского

В последнее время в качестве высокопроизводительных средств для решения научных и производственных задач большой вычислительной сложности все чаще применяются кластерные системы [1,2,3]. Обусловлено это, в первую очередь, тем, что кластеры при мощности, сопоставимой с суперкомпьютерами, имеют гораздо лучшее соотношение цена/производительность.

Поскольку вычислительная стоимость передачи данных (коммуникации между узлами) в кластерах в десятки, а то и сотни раз выше чем в специально спроектированных суперкомпьютерах, большую роль приобретают, во-первых, грамотный выбор топологии сети, а во-вторых, оптимальное распределение подзадач по узлам, от которого во многом зависит производительность кластера при решении задачи в целом. Последняя задача решается с помощью специальных программных системам управления кластерами. В качестве примера таких систем можно привести широко распространенные продукты CCS, LSF, OpenPBS и другие [4,5,6].

Важными частями практически любой программы управления кластером являются монитор, собирающий информацию о доступности и загруженности узлов вычислительной системы, и планировщик (scheduler), на основании поступающей информации производящий постановку пользовательского задания в очередь на выполнение. От точности данных, поступающих от монитора и реализованного в планировщике алгоритма диспетчеризации во многом зависит производительность кластера, а, значит, и его способность решать сложные и объемные вычислительные задачи.

Мониторинг узлов кластера – это действие по сбору информации о некоторых параметрах узлов системы, таких как загрузка центрального процессора, наличие свободной памяти, скорость ввода/вывода и т.д. и представление их в удобной для восприятия форме пользователю или администратору системы. Мониторинг является важным условием стабильной работы больших кластерных систем, поскольку позволяет администратору (или управляющей

программе) обнаруживать и устранять потенциальные проблемы, связанные с производительностью и функционированием кластера. Помимо этого, с помощью мониторинга другие компоненты управляющих программ (например, планировщик) могут улучшить свою работу, обеспечивая более эффективную балансировку загрузки системы.

Большинство средств мониторинга состоит из нескольких подсистем и организовано следующим образом (рис.1) [1]:

- Модуль сбора информации о производительности (Performance Probing Module, PPM): часть системы мониторинга, служащая для сбора информации на узле и взаимодействующая с операционной системой. Полученные данные передаются в другие подсистемы мониторинга. В некоторых реализациях [7] PPM также производит проверки некоторых событий на узле (например, изменение содержимого файлов) и генерирует уведомление, если эти события произошли.
- Модуль накопления информации о производительности (Performance Collection Module, PCM): эта часть системы мониторинга собирает информацию с узлов для последующей обработки. Одной из главных его задач является кэширование информации для снижения влияния процесса мониторинга на производительность системы.
- Библиотека интерфейсных функций для доступа к информации о производительности (Performance API Library Module, PAM): позволяет программисту получить в своем приложении доступ к информации о производительности.
- Модуль контроля и визуализации (The Control and Visualization Module, CVM): приложение, которое предоставляет информацию о производительности пользователю (администратору) в наглядном виде (текстовом или графическом). Другой важной функцией является предоставление пользователю контроля над состоянием узлов и различные варианты фильтрации получаемой информации.

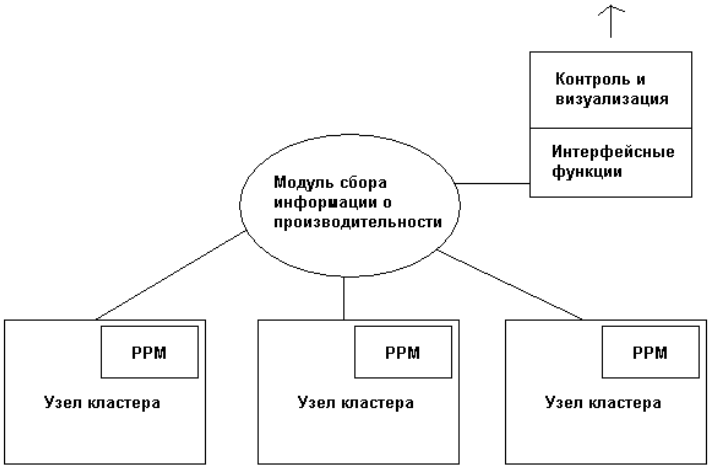


Рис.1

Операционные системы семейства Microsoft Windows NT предоставляют интерфейс Windows Management Instrumentation (WMI) [8] для получения системной информации. Достоинством этого подхода является легкость использования и независимость от ядра ОС. Существует несколько аналогов этого API, разработанных под UNIX-системы, однако ни один из них не реализован настолько полно как WMI.

Сбор получаемой информации может производиться как распределенно (и затем собираться специальным модулем накопления информации), так и централизованно [9]. Централизованный подход хорошо работает на относительно небольших кластерах – от 16 до 64 узлов. Однако, для кластерных систем, имеющих более 250 узлов, такой подход становится неадекватным, поскольку время сбора информации увеличивается настолько, что к моменту получения она становится бесполезной. Одним из способов избежать этой проблемы является иерархическое построение систем сбора информации [10]. Множество узлов разделяется на домены, каждый их которых имеет свой прокси-монитор. Прокси-монитор получает запросы от вышестоящих доменов, собирает, объединяет и передает им

информацию, реагируя и передавая на верхний уровень также данные о сбоях в своем домене.

Важной задачей при проведении мониторинга является минимизация влияния монитора на общую производительность системы, в частности, уменьшение сетевого траффика. Существует несколько способов снижения сетевой активности средств мониторинга:

- Выбор частоты опроса. Целью опроса узлов является получение текущего состояния кластера в какой-то момент времени. При этом точность отображения этого состояния для пользователя зависит от частоты, с которой данные снимаются с узлов. Более точное представление данных ведет к увеличению частоты опроса, и, следовательно, к увеличению сетевого траффика, что нежелательно. В некоторых системах возможно задание частоты опроса, что позволяет регулировать как точность отображения информации о состоянии кластера, так и объем трафика. Если информация о системе используется только для планировщика, то возможен опрос «on demand», то есть, непосредственно в момент запуска задачи планировщиком. Также можно комбинировать оба способа, например, установив регулярный опрос с большим периодом и дополнительно запрашивая данные при поступлении очередной задачи.
- Определение набора собираемых данных. В большинстве случаев администратору (или планировщику) требуется ограниченный объем данных о параметрах узлов кластера. Поэтому, предоставление строго необходимого набора параметров может значительно уменьшить объем передаваемой по сети информации. Как правило, возможность фильтровать информацию и определять ее объем предоставляется API.
- Для иерархических систем мониторинга – объединение данных. Данные от доменов нижнего уровня объединяются и представляются в едином виде. Таким образом, избыточная информация не передается, что снижает траффик.

Еще один важный аспект работы средств мониторинга – минимальная интрузивность (влияние на работу остальных компонент системы управления и пользовательских задач). Время сбора инфор-

мации о производительности и пересылки ее необходимым пользователям не должно быть слишком большим. Также сбор данных не должен занимать много ресурсов на опрашиваемых узлах, влияя тем самым на производительность других задач на этих компьютерах. Часто уменьшение интрузивности (например, с помощью использования особенностей операционной системы) приводит к потере переносимости кода монитора [11]. В случае разработки многоплатформенной системы мониторинга необходимо принимать это во внимание. С другой стороны, интрузивность можно уменьшить, подобрав соответствующую частоту опроса машин и набор запрашиваемых данных.

- Внедряемые в последнее время web-технологии предоставляют множество преимуществ пользователям систем мониторинга:
- доступ к данным о производительности через Интернет,
- удобный пользовательский интерфейс на основе возможностей интернет-браузера,
- возможность получать информацию о состоянии кластера через низкоскоростные каналы.

Пример типичной системы мониторинга, основанной на web-технологии, приведен на рис.2. [12].

Прием и обработка информации от web-сервера может производиться как с помощью Java, так и другими средствами, например, с использованием Microsoft .NET. Однако, во всех случаях необходимо обеспечить надлежащую безопасность web-системы мониторинга и разграничение прав и доступа пользователей к ней.

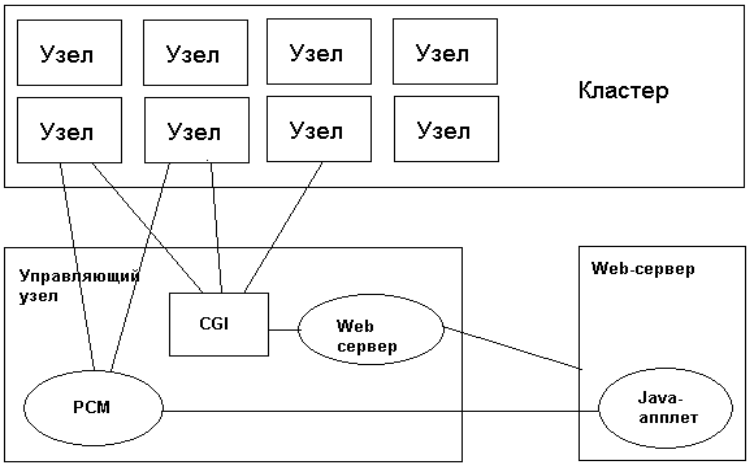


Рис.2

Литература

1. Baker M. et al. Cluster Computing White Paper. Final Release, Version 2.0.
2. IEEE Task Force on Cluster Computing. http://www.ieeetfcc.org/
3. Учебно-информационный центр Parallel.ru. http://parallel.ru
4. Keller A., Reinfeld A. Anatomy of a Resource Management System for HPC Clusters. Preprint. To appear in: Annual Review of Scalable Computing, Vol. 3, 2001.
5. Platform LSF 5. http://www.platform.com/products/wm/LSF/index.asp
6. OpenPBS home page. http://www.openpbs.org
7. RSYNC, http://rsync.samba.org.
8. Monitoring Cluster Computers With WMI. Cornell Theory Center, virtual workshop module.
9. VA Cluster Manager. http://vacm.sourceforge.net
10. Liang Z., Sun Y. and Wang C. ClusterProbe: An Open, Flexible and Scalable Cluster Monitoring Tool, in IWCC99: Preceedings of international workshop on cluster computing 99, Australia, 1999.
11. P.Uthayopas, S.Phatanapherom. Fast and Scalable Real-time Monitoring System for Beowulf Clusters . Lecture Notes in Computer Science.

12. KCAP: Kasetsart Cluster Administration Program, http://amata.cpe.ku.ac.th/kcap.

МОДЕЛИРОВАНИЕ И ПРОГНОЗ АТМОСФЕРНЫХ ПРОЦЕССОВ ДЛЯ РЕГИОНА

Г. Лучицкий

grishalu@uninet.kiev.ua

Численное моделирование атмосферных процессов является важной частью в прогнозировании погоды, а также в прогнозировании природных катаклизмов. С развитием вычислительной техники численное моделирование атмосферных процессов пережило настоящий бум, но для получения по-настоящему точных результатов надо решать задачи моделирования атмосферных процессов на очень мелкой сетке, а это, в свою очередь, требует громадных вычислительных ресурсов. Поэтому очень перспективным решением данной проблемы есть расщепление задачи на более простые, которые вычисляются параллельно.

Рассмотрим модель атмосферных процессов для региона. Определим связь компонентов u, v, w скорости $\mathbf{V}$ с компонентами сферической системы координат λ, φ, z следующим образом: u – в направлении возрастания λ (направление на «восток»), v – в направлении возрастания φ (направление на «север»), w – в направлении возрастания z (направление, противоположное гравитационной силе, то есть в «верх»). Пусть также

$$\sigma = \frac{z - F(\lambda, \varphi)}{H - F(\lambda, \varphi)},$$

$$\overline{w} = \frac{1}{H - F}\left[w - (1 - \sigma)\left(\frac{u}{r\cos\varphi}\frac{\partial F}{\partial \lambda} + \frac{v}{r}\frac{\partial F}{\partial \varphi} \right) \right],$$

где λ – долгота; φ – широта; σ – приведенная вертикальная координата, которая отслеживает рельеф; F – высота рельефа; H – высота верхней границы области решения задачи над уровнем моря.

Региональную модель атмосферы представим с помощью следующей системы уравнений:

$$\frac{\partial u}{\partial t} = -\frac{u}{r\cos\varphi}\frac{\partial u}{\partial \lambda} - \frac{v}{r}\frac{\partial u}{\partial \varphi} - \overline{w}\frac{\partial u}{\partial \sigma} + \left(2\Omega + \frac{u}{r\cos\varphi}\right)v\sin\varphi -$$
$$-\frac{1}{r\cos\varphi}\left[\theta_v \frac{\partial \pi}{\partial \lambda} + (1-\sigma)g\frac{\partial F}{\partial \lambda}\right] + \frac{1}{r\cos\varphi}\frac{\partial}{\partial \lambda}\left(K_G \frac{\partial u}{\partial \lambda}\right) + \quad (1)$$
$$+\frac{1}{r}\frac{\partial}{\partial \varphi}\left(K_G \frac{\partial u}{\partial \varphi}\right) + \frac{1}{(H-F)^2}\frac{\partial}{\partial \sigma}\left(K_M \frac{\partial u}{\partial \sigma}\right)$$

$$\frac{\partial v}{\partial t} = -\frac{u}{r\cos\varphi}\frac{\partial v}{\partial \lambda} - \frac{v}{r}\frac{\partial v}{\partial \varphi} - \overline{w}\frac{\partial v}{\partial \sigma} + \left(2\Omega + \frac{u}{r\cos\varphi}\right)u\sin\varphi -$$
$$-\frac{1}{r}\left[\theta_v \frac{\partial \pi}{\partial \varphi} + (1-\sigma)g\frac{\partial F}{\partial \varphi}\right] + \frac{1}{r\cos\varphi}\frac{\partial}{\partial \lambda}\left(K_G \frac{\partial v}{\partial \lambda}\right) + \quad (2)$$
$$+\frac{1}{r}\frac{\partial}{\partial \varphi}\left(K_G \frac{\partial v}{\partial \varphi}\right) + \frac{1}{(H-F)^2}\frac{\partial}{\partial \sigma}\left(K_M \frac{\partial v}{\partial \sigma}\right)$$

$$\frac{1}{(H-F)^2}\frac{\partial^2 w}{\partial \sigma^2} + \frac{1}{H-F}\left(\frac{2}{r} - \frac{g}{\pi\theta_V}\right)\frac{\partial w}{\partial \sigma} - \frac{2}{r}\left(\frac{1}{r} + \frac{g}{\pi\theta_V}\right)w =$$
$$= \frac{1}{r\cos\varphi}\left\{\left(\frac{1}{r} + \frac{g}{\pi\theta_V}\right)\left[\left(\frac{\partial u}{\partial \lambda} + \frac{\partial v\cos\varphi}{\partial \varphi}\right) - \frac{1}{H-F}\left(\frac{\partial F}{\partial \lambda}\frac{\partial u}{\partial \sigma} + \frac{\partial F}{\partial \varphi}\frac{\partial v\cos\varphi}{\partial \sigma}\right)\right] - \right. \quad (3)$$
$$\left. -\frac{1}{H-F}\frac{\partial}{\partial \sigma}\left[\frac{\partial u}{\partial \lambda} + \frac{\partial v\cos\varphi}{\partial \varphi} - \frac{1-\sigma}{H-F}\left(\frac{\partial F}{\partial \lambda}\frac{\partial u}{\partial \sigma} + \frac{\partial F}{\partial \varphi}\frac{\partial v\cos\varphi}{\partial \sigma}\right)\right]\right\}$$

$$\frac{\partial \theta}{\partial t} = -\frac{u}{r\cos\varphi}\frac{\partial \theta}{\partial \lambda} - \frac{v}{r}\frac{\partial \theta}{\partial \varphi} - \overline{w}\frac{\partial \theta}{\partial \sigma} +$$
$$+\frac{1}{r\cos\varphi}\frac{\partial}{\partial \lambda}\left(K_G \frac{\partial \theta}{\partial \lambda}\right) + \frac{1}{r}\frac{\partial}{\partial \varphi}\left(K_G \frac{\partial \theta}{\partial \varphi}\right) + \frac{1}{(H-F)^2}\frac{\partial}{\partial \sigma}\left(K_H \frac{\partial \theta}{\partial \sigma}\right) - \quad (4)$$
$$-\frac{L}{\pi}\left(\delta\frac{dq_H}{dt}\right) + Q_K - Q_I + Q_R$$

$$\frac{\partial q}{\partial t} = -\frac{u}{r\cos\varphi}\frac{\partial q}{\partial \lambda} - \frac{v}{r}\frac{\partial q}{\partial \varphi} - \overline{w}\frac{\partial q}{\partial \sigma} +$$
$$+\frac{1}{r\cos\varphi}\frac{\partial}{\partial \lambda}\left(K_G\frac{\partial q}{\partial \lambda}\right) + \frac{1}{r}\frac{\partial}{\partial \varphi}\left(K_G\frac{\partial q}{\partial \varphi}\right) + \frac{1}{(H-F)^2}\frac{\partial}{\partial \sigma}\left(K_H\frac{\partial q}{\partial \sigma}\right) + \quad (5)$$
$$+\delta\frac{dq_H}{dt} + M_K - M_I$$

$$\frac{\partial q_L}{\partial t} = -\frac{u}{r\cos\varphi}\frac{\partial q_L}{\partial \lambda} - \frac{v}{r}\frac{\partial q_L}{\partial \varphi} - \overline{w}\frac{\partial q_L}{\partial \sigma} + \quad + \frac{1}{r\cos\varphi}\frac{\partial}{\partial \lambda}\left(K_G\frac{\partial q_L}{\partial \lambda}\right) +$$
$$+\frac{1}{r}\frac{\partial}{\partial \varphi}\left(K_G\frac{\partial q_L}{\partial \varphi}\right) + \frac{1}{(H-F)^2}\frac{\partial}{\partial \sigma}\left(K_H\frac{\partial q_L}{\partial \sigma}\right) + \quad (6)$$
$$+\frac{1}{\rho(H-F)}\frac{\partial \rho V_o q_L}{\partial \sigma} - \delta\frac{dq_H}{dt} - M_I$$

$$\frac{\partial \pi}{\partial \sigma} = -\frac{g(H-F)}{\theta_v}, \quad (7)$$

$$\frac{\partial p}{\partial t} = g\rho\overline{w} - 2(g\rho\overline{w})_{\sigma=1} - g(H-F)\int_{\sigma}^{1}\frac{1}{r\cos\varphi}\left(\frac{\partial \rho u}{\partial \lambda} + \frac{\partial \rho v\cos\varphi}{\partial \varphi}\right)d\zeta\ , \quad (8)$$

$$\frac{\partial k}{\partial t} = -\frac{u}{r\cos\varphi}\frac{\partial k}{\partial \lambda} - \frac{v}{r}\frac{\partial k}{\partial \varphi} - \overline{w}\frac{\partial k}{\partial \sigma} + \frac{K_M}{(H-F)^2}\left[\left(\frac{\partial u}{\partial \sigma}\right)^2 + \left(\frac{\partial v}{\partial \sigma}\right)^2\right] -$$
$$-\frac{g}{\theta_v}\frac{K_H}{H-F}\frac{\partial \theta_v}{\partial \sigma} + \frac{2}{(H-F)^2}\frac{\partial}{\partial \sigma}\left(K_M\frac{\partial k}{\partial \sigma}\right) - \varepsilon \quad (9)$$

$$\frac{\partial \varepsilon}{\partial t} = -\frac{u}{r\cos\varphi}\frac{\partial \varepsilon}{\partial \lambda} - \frac{v}{r}\frac{\partial \varepsilon}{\partial \varphi} - \overline{w}\frac{\partial \varepsilon}{\partial \sigma} +$$

$$+ C_2 \frac{\varepsilon}{k}\left\{\frac{K_M}{(H-F)^2}\left[\left(\frac{\partial u}{\partial \sigma}\right)^2 + \left(\frac{\partial v}{\partial \sigma}\right)^2\right] - \frac{g}{\theta_v}\frac{K_H}{H-F}\frac{\partial \theta_v}{\partial \sigma}\right\} - \qquad (10)$$

$$- C_3 \frac{\varepsilon^2}{k} + \frac{C_4}{(H-F)^2}\frac{\partial}{\partial \sigma}\left(K_M \frac{\partial \varepsilon}{\partial \sigma}\right)$$

$$K_M = C_1 k^2 / \varepsilon . \qquad (11)$$

где t – время; r – радиус Земли; Ω – угловая скорость вращения Земли; $\pi = C_p (p/p_0)^{R/C_p}$ – приведенное давление; p – давление, p_0 – значение давления на поверхности моря; ρ – плотность воздуха; $g = 9{,}81$ – ускорение свободного падения; T – температура, θ – температура потенциального потока, θ_v – виртуальная температура потенциального потока; q – удельная влажность, $q_н$ – влажность в состоянии насыщения; q_L – удельная водность; $R = 287{,}04$ – газовая постоянная воздуха; $C_p = 3{,}5R$ – теплоемкость воздуха при условии постоянного давления; L – скрытое тепло конденсации; δ – признак наличия конденсации влажность (1 – имеет место, 0 – отсутствует); Q_K – интенсивность высвобождения скрытой теплоты конденсации паров воды в подсеточном масштабе; Q_I – интенсивность высвобождения скрытой теплоты испарения воды в подсеточном масштабе; Q_R – радиационное охлаждения или нагрев; M_K– источнник влажности от конденсации в подсеточном масштабе; M_I – источнник влажности от испарения в подсеточном масштабе; V_o – установившаяся скорость осадков; k – турбулентная кинетическая энергия; ε – турбулентная диссипация; K_G – коэффициент горизонтального турбулентного обмена; K_M – коэффициент вертикального турбулентного обмена для количества движения; K_H – коэффициент вертикального турбулентного обмена для тепла и влажности; $(C_1, C_2, C_3, C_4) = (0{,}09; 1{,}46; 1{,}83, 0{,}42)$ – константы замыкания пограничного слоя атмосфери.

Математическая модель (1)–(11) отличается от известных и широко используемых в оперативной практике моделей применением вертикальной координаты σ и уравнениями (3), (7), (8).

Допустим, что состояние атмосферы в фазовом пространстве $r = (\lambda, \varphi, \sigma)$ определяется вектором $\Re(r,t) = (u, v, \pi, T, q, q_L, k, \varepsilon)$, который задан в макромасштабной области $G = G(r)$. Пусть в области G заданы $\Re(r, t^m) = \Re^m(r)$ в моменты времени $t = t^m$ $(m = 0,1,...,M)$. Тогда состояние атмосферы на ограниченной территории $\overline{G}$ для $\forall t \in [t^m, t^{m+1}]$ можно получить, решая региональную задачу

$$\frac{\partial \Re}{\partial t} = D\Re\ ,\ \forall t \in S\ ,\ \forall r \in \overline{G}\ , \qquad (12)$$

$$\Re\left(r, t^m\right) = \Re^m(r)\ ,\ m = 0,1,...,M$$

Численная реализация макромасштабных гидродинамических моделей (планетарных или полусферных), как правило, выполняется на дискретном множестве точек, горизонтальные расстояния между которыми на один – два порядка больше, нежели на разностных сетках, которые используются при численном решении гидродинамических моделей регионального масштаба (12). Следовательно, использование решений макромасштабных задач в качестве краевых условий для региональной модели (12) требует применения вложенных сеток и метода «одностороннего воздействия».

Рассмотрим независимую непрерывную переменную $r = (\lambda, \varphi, \sigma)$ в ограниченной области $\overline{G} = \overline{G}(r)$. Заменим континуум пространственной сеткой из точек путем разбивки области $\overline{G}$ на множество из $J - 1$ элементов $\Delta\lambda_j$, $K - 1$ элементов $\Delta\varphi_k$ и $L - 1$ элементов $\Delta\sigma_l$. Построим вектор $\{r_{jkl}\}$, определяя непрерывную переменную r только в точках j $(1 \le j \le J)$, k $(1 \le k \le K)$, l $(1 \le l \le L)$. В результате получим:

$$\lambda_J = \lambda_1 + \sum_{\mu=2}^{J-1} \Delta\lambda_\mu\ ,\ \varphi_K = \varphi_1 + \sum_{\mu=2}^{K-1} \Delta\varphi_\mu\ ,\ \sigma_L = \sigma_1 + \sum_{\mu=2}^{L-1} \Delta\sigma_\mu\ . \qquad (13)$$

В области определения $R = \overline{G} \times S$ вместо функции $\Re(r,t)$, заданной на макромасштабной сетке, будем искать функцию дискретного аргумента $\Re\left(r_{jkl}, t^m\right) = \Re^m_{jkl}$ на региональной сетке. Значения этой функции будем искать в узлах сетки $(\lambda_j, \varphi_k, \sigma_l, t^m) \in R$, j $(1 \le j \le J)$, k $(1 \le k \le K)$, l $(1 \le l \le L)$, m $(1 \le m \le M)$. Кроме того

дифференциальному оператору D в (12) поставим в соответствие сеточный оператор Λ, который удовлетворяет главным требованиям аппроксимации: заданному порядку точности, сходимости $\Lambda\Re \to D\Re$ (при $h \to 0$, где $h = (\Delta\lambda_j, \Delta\varphi_k, \Delta\sigma_l)$ – шаг пространственной сетки), консервативности (выполнение интегральных законов сохранения, справедливых для исходного дифференциального оператора D), транспортивности (возмущение, наложенное на функцию $\Re$, переносится за счет конвекции только в направления скорости). Поставленную задачу восполнения значений функции $\Re(r, t)$, заданной на макромасштабнгой сетке, в узлы региональной сетки и вычисление сеточных значений $f^m_{jkl} = \Lambda\Re^m_{jkl}$ будем решать в комплексе.

После проведения операции восполнения функции $\Re(t^m) = \Re^m$ в узлы региональной сетки и вычисления значений правой части $f(t^m) = f^m$, $m = 1,2,\ldots, M$ решение задачи (12) ищется для $\forall t \in [t^m, t^{m+1}]$ по формуле

$$\Re(t) = \Re^m + \frac{t-t^m}{\tau}\left[\ \tau f^m + \frac{t-t^m}{\tau}\left[\ \Re^{m+1} - 2\Re^m + \Re^{m-1} - \right.\right.$$

$$-\frac{\tau}{4}\left(f^{m+1} - f^{m-1}\right) + \frac{t-t^m}{4\tau}\left[\ 5\left(\Re^{m+1} - \Re^{m-1}\right) - \tau\left(f^{m+1} + 8f^m + f^{m-1}\right) - \right. \tag{13}$$

$$-\frac{t-t^m}{\tau}\left[\ 2\left(\Re^{m+1} - 2\Re^m + \Re^{m-1}\right) - \tau\left(f^{m+1} - f^{m-1}\right) + \right.$$

$$\left.\left.\left.\left.\left.+\frac{t-t^m}{4\tau}\left[\ 3\left(\Re^{m+1} - \Re^{m-1}\right) - \tau\left(f^{m+1} + 8f^m + f^{m-1}\right)\right]\right]\right]\right]\right]$$

для каждого узла сетки $(\lambda_j, \varphi_k, \sigma_l)$, j $(1 \le j \le J)$, k $(1 \le k \le K)$, l $(1 \le l \le L)$.

Схема (13), как легко проверить, обладает интерполирующими свойствами, т.е. при $t = t^m$ или $(\tau = t - t^m = 0)$ и $t = t^{m+1}$ или $(\tau = t^{m+1} - t = 0)$ выполняются равенства $\Re(t^m) = \Re^m$ и $\Re(t^{m+1}) = \Re^{m+1}$ соответственно. Следовательно, максимальная ошибка решения задачи (12) с помощью (13) находится внутри отрезка $t^m \le t \le t^{m+1}$ и определяется порядком аппроксимации, т.е. равна $O[(\tau^4)]$.

Симметризация исходных уравнений и предложенныя разностная схема естественным образом позволяют использовать при решении полученной нестационарной задачи метод расщепления, сводящий

решение сложной трехмерной задачи к более простым, не нарушая при этом квадратичных законов сохранения и обеспечивая хорошую точность. По предварительным расчетам, в результате расщепления задачи и распараллеливания вычислений более простых задач, может достигатся увеличение скорости решения задачи в 27 раз.

Литература

1. Андерсон Д., Танненхил Дж., Плетчер Р. Вычислительная гидромеханика и теплообмен: В 2-х т. Т.1: Пер. с англ. – М.: Мир, 1990.
2. Белов П.Н., Борисенков Е.П., Панин Б.Д. Численные методы прогноза погоды – Л.: Гидрометеоиздат, 1989. – 375с.
3. Довгий С.А., Прусов В.А., Копейка О.В. Математическое моделирование техногенных загрязнений окружающей среды – К.: Наукова Думка, 2000, – 284 с.
4. Дорошенко А.Е. Математические модели и методы организации высокопроизводительных параллельных вычислений. Алгебродинамический подход.- Киев:Наукова думка, 2000.- 177 с.
5. Марчук Г.И., Дымников В.П., Залесный В.Б. Математическое моделирование общей циркуляции атмосферы и океана – Л.: Гидрометеоиздат, 1984. – 320 с.

ПРОГРАММНАЯ СИСТЕМА ИССЛЕДОВАНИЯ ЗАДАЧ АППРОКСИМАЦИИ СМЕШАННОГО ТИПА

В.С.Орлов

Нижегородский государственный университет им.Н.И.Лобачевского

Введение

Существуют различные задачи, которые можно представить как задачи оптимизации. Например, задачи аппроксимации, возникающие при проектировании радиотехнических устройств, отыскании решения систем нелинейных уравнений, могут быть сведены к минимизации некоторой невязки, которая часто оказывается многоэкстремальной.

Далее мы рассмотрим задачу смешанной аппроксимации, сводящуюся к задаче оптимизации, и ее практическую реализацию в виде программной системы исследования задач такого класса.

Постановка задачи

Для практического исследования ставится задача смешанной аппроксимации таблично заданной функции, т.е. задача нахождения представителя параметрически заданного класса А, состоящего из функций

$$\varphi(a,b,x) = \sum_{i=1}^{k} a_i \varphi_i(b,x); \qquad (1)$$

где $a = (a_1,\ldots,\ a_k)$, $b = (b_1,\ldots,\ b_m)$ – параметры; причем a_i входят в $\varphi(a, b, x)$ линейно, а b_i – не линейно.

В качестве меры рассогласования рассматривается невязка вида

$$N(a,b) = \sum_{i=0}^{N} (\varphi(x_i,a,b) - y_i)^2, \qquad (2)$$

где (x_i, y_i) – таблично заданные точки, $0 \le i \le N$.

Окончательно задача выбора параметров ставится как задача оптимизации

$$N^* = \min_{b\in B} \min_{a\in R^k} N(a,b) \qquad (3)$$

Алгоритм решения задачи оптимизации

Для решения задачи (3) используется многомерный обобщенный алгоритм глобального поиска с неинъективной разверткой [1], с помощью которого вычисляются конкретные $\tilde{b} \in B$. Надо заметить, что при решении данной задачи на многопроцессорных системах нужно использовать параллельный алгоритм глобальной оптимизации [5]. Потом при каждом таком фиксированном $\tilde{b} \in B$ решается следующая задача:

$$N^*_{\tilde{b}} = \min_{a\in R^k} N(a,\tilde{b}), \qquad (4)$$

где

$$N(a,\tilde{b}) = \sum_{i=0}^{N} (\varphi(x_i,a,\tilde{b}) - y_i)^2, \qquad (4.1)$$

$$\varphi(x,a,\tilde{b}) = \sum_{i=1}^{k} a_i \varphi_i(x,\tilde{b}),\ \ a = (a_1,\ldots,a_k) \in R^k, \qquad (4.2)$$

а (x_i, y_i) – таблично заданные точки, $0 \le i \le N$.

Для решения задачи (4) используется метод наименьших квадратов [2].

В соответствии с методом наименьших квадратов получим неоднородную систему линейных алгебраических уравнений:

$$Ma^T = c^T, \tag{5}$$

где $a = (a_1, \ldots, a_k) \in R^k$,

$c = (c_1, \ldots, c_k) \in R^k$, $c_j = \sum_{i=0}^{N} \varphi_j(x_i, \tilde{b}) y_i; j = 1, \ldots, k;$

$M = (m_{jn})_{kxk}$, $m_{jn} = \sum \varphi_n(x_i, \tilde{b}) \varphi_j(x_i, \tilde{b}); \quad j = 1, \ldots, k; n = 1, \ldots, k.$

Таким образом, решение задачи (4) сводится к решению системы линейных алгебраических уравнений (5).

Для решения системы (5) и нахождения $a = (a_1, \ldots, a_k) \in R^k$ можно использовать метод Гаусса с выбором главного элемента [3], либо один из итерационных методов, к примеру, метод Зейделя [6].

Вычислительные эксперименты

Для проведения вычислительных экспериментов в указанном классе задач была создана программная система, обеспечивающая решение задачи (3). Программная система реализована в среде Microsoft Visual Studio.NET на языке программирования C++ с использованием библиотеки классов MFC (Microsoft Foundation Class) [4].

Созданная программная система обеспечивает решение задачи (3) при *k* линейно входящих параметрах и при *m* нелинейно входящих параметрах, используя описанный выше алгоритм. Практически размерность задачи ограничена ресурсами используемого компьютера.

Данная программная система создана под Windows и реализует дружественный диалоговый интерфейс для задания исходных параметров.

Реализованы диалоги для ввода следующих параметров:

- функций с использованием формульного транслятора, что позволяет их вводить в привычном для пользователя виде;
- диапазонов изменения нелинейно входящих параметров;
- точности и максимального количества итераций;

- ввод координат точек для табличного задания функции;
- области просмотра координатной плоскости;

Также реализована возможность табличного задания функции путем отметки точек с помощью «мыши» на координатной плоскости.

Система позволяет отображать полученные результаты в графическом виде:

- отображение графика полученной функции на координатной плоскости с возможностью выбора области просмотра с помощью «мыши».
- отображение введенных функций и полученных после вычислений параметров в дереве просмотра.

После вычислений статистические данные работы алгоритма отображаются в диалоге результатов.

Также реализована возможность сохранения заданных и полученных параметров во внешнем файле на диске и загрузка сохраненного задания из файла.

Заключение

Программная система представляет собой готовый к применению программный продукт и может быть использована как в учебном процессе при изучении методов оптимизации, так и при решении прикладных задач аппроксимации.

Использование же параллельных методов вычислений может существенно ускорить вычисления, соответствующих одновременному определению значения оптимизируемой функции в нескольких точках области определения, и являются перспективным направлением исследования.

Пример

$$\varphi(x,a,b) = a_1(x+b_1)^4 + a_2(x+b_2)^3 + a_3(x+b_3)^2 + a_4 + \\ + a_5\sin(b_4x) + a_6\cos(b_5x)$$

$$b_1 \in [-100;100], b_2 \in [-100;0], b_3 \in [-100;0], b_4 \in [-100;100],$$

$$b_5 \in [0;100] r = 2, \varepsilon = 0.00001$$

Задано 24 точки.
Размерность задачи $N = 5$.

Максимальное количество итераций = 1000.

Результаты:
Невязка N^*= 0.079876315553973787
Максимальное отклонение = 0.14108326745504
Было проделано 1000 итераций. Заданная точность не была достигнута за данное количество итераций.
Полученные коэффициенты:
a_1=0.000017295694372152
a_2=-0.014768003524118280
a_3=-4.634255701566933
a_4=-6562.3657356183876
a_5=1.23540311720519
a_6=-0.000237613339192939
b_1=89.64996337890625
b_2=-88.135528564453125
b_3=-31.737518310546875
b_4=1.56097412109375
b_5=18.604278564453125

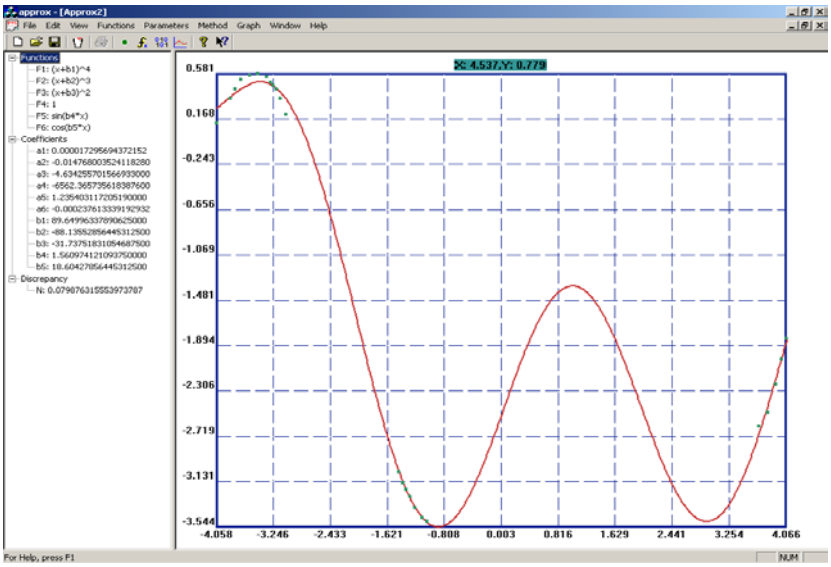


Рис.1

Литература

1. Стронгин Р.Г. Численные методы в многоэкстремальных задачах (информационно-статистические алгоритмы). – М.: Наука, 1978, 240с.
2. Демидович Б.П., Марон И.А., Шувалова Э.З. Численные методы анализа. – М: Физматгиз, 1963, 400с.
3. Волков Е.А. Численные методы: Учеб. пособие для вузов. – М.: Наука. 1987, – 248с.
4. Мешков А.В., Тихомиров Ю.В. Visual C++ и MFC: Пер. с англ. – СПб.: БХВ-Петербург, 2002. – 1040с.
5. Сергеев Я.Д., Стронгин Р.Г., Гришагин В.А. Введение в параллельную глобальную оптимизацию. Учебное пособие. Нижний Новгород: Изд-во Нижегородского университета, 1998. 87 с.
6. Бахвалов Н.С. Численные методы (анализ, алгебра, обыкновенные дифференциальные уравнения). – М.: Наука, 1975.

ПРИМЕНЕНИЕ КЛАСТЕРНЫХ СИСТЕМ ДЛЯ МОДЕЛИРОВАНИЯ ДВИЖЕНИЯ КОСМИЧЕСКОГО АППАРАТА

Н.Д. Пиза, Р.К. Кудерметов

Запорожский национальный технический университет, Украина
ndp@zntu.edu.ua, krk@zntu.edu.ua

Введение

При наземной отработке и испытаниях систем управления космических аппаратов (КА) в составе полунатурных моделирующих комплексов применяется моделирование полета КА в реальном и ускоренном масштабах времени. Моделирование состоит в численном решении дифференциальных и алгебраических уравнений, описывающих движение центра масс КА, движение его вокруг собственного центра масс, возмущающие и управляющие моменты, действующие на КА. Для некоторых этапов полунатурной отработки необходимы более полные и сложные модели, что требует применения высокопроизводительных вычислительных систем. Представляет

интерес исследовать возможность использования для этих целей параллельных вычислительных систем.

Значительную долю вычислений при моделировании движения КА занимают вычисления, связанные с интегрированием уравнений движения. В данной статье приведены результаты исследования характеристик параллельных блочных методов интегрирования, реализованных на языке С с использованием функций библиотеки MPI (реализация mpich 1.2.4) при моделировании движения КА на кластерных вычислительных системах.

Параллельные блочные одношаговые методы интегрирования

Для задачи Коши для системы обыкновенных дифференциальных уравнений

$$\frac{dy}{dt} = f(t, y),\ t > 0,\ y(0) = y_0 \tag{1}$$

параллельный блочный двухточечный одношаговый метод четвертого порядка точности $O(h^4)$ на равномерной сетке по переменной t с шагом $h > 0$ реализуется следующими рекуррентными формулами [1]:

$$\begin{aligned} y_{n+r,0} &= y_n + rhf_n,\ r = 1, 2 \\ y_{n+1,s+1} &= y_n + \frac{h}{12}\left(5f_n + 8f_{n+1,s} - f_{n+2,s}\right) \\ y_{n+2,s+1} &= y_n + \frac{h}{3}\left(f_n + 4f_{n+1,s} + f_{n+2,s}\right)\quad s = \overline{0, 2} \end{aligned} \tag{2}$$

Как видно из структуры приведенных формул, вычисления значений функций $f_{n+1,\,s}$, $f_{n+2,\,s}$ и получаемых решений $y_{n+1,\,s+1}$, $y_{n+2,\,s+1}$, можно выполнять параллельно. Таким образом, решение в n-м блоке получается сразу в двух точках: n_1, n_2 – это, соответственно, $y_{1,3}$, $y_{2,3}$, причем, значение $y_{2,3}$ является начальным для следующей итерации, т.е. выступает в качестве y в $n + 1$-м блоке.

Рекуррентные формулы для параллельного блочного четырехточечного одношагового метода шестого порядка точности $O(h^6)$ имеют вид [1]:

$$y_{n+r,0} = y_n + rhf_n,\ r = \overline{1,4},$$

$$
\begin{aligned}
y_{n+1,s+1} &= y_n + \frac{h}{720}\left(251 f_n + 646 f_{n+1,s} - 264 f_{n+2,s} + 106 f_{n+3,s} - 19 f_{n+4,s}\right) \\
y_{n+2,s+1} &= y_n + \frac{h}{90}\left(29 f_n + 124 f_{n+1,s} + 2 f_{n+2,s} + 4 f_{n+3,s} - f_{n+4,s}\right) \\
y_{n+3,s+1} &= y_n + \frac{3h}{80}\left(9 f_n + 34 f_{n+1,s} + 24 f_{n+2,s} + 14 f_{n+3,s} - f_{n+4,s}\right) \\
y_{n+4,s+1} &= y_n + \frac{2h}{45}\left(7 f_n + 32 f_{n+1,s} + 12 f_{n+2,s} + 32 f_{n+3,s} + 7 f_{n+4,s}\right), \quad s = \overline{0,4}
\end{aligned}
\qquad (2')
$$

Здесь вычисления функций $f_{n+1,\,s}$, $f_{n+2,\,s}$ $f_{n+3,\,s}$, $f_{n+4,\,s}$ и получаемых решений $y_{n+1,\,s+1}$, $y_{n+2,\,s+1}$, $y_{n+3,\,s+1}$, $y_{n+4,\,s+1}$ можно выполнять параллельно. В данном методе решения для n -го блока получаются одновременно в четырех точках: n_1, n_2, n_3, n_4 – это, соответственно, $y_{1,5}$, $y_{2,5}$, $y_{3,5}$, $y_{4,5}$, причем, значение $y_{4,5}$ является начальным для вычислений в следующем блоке.

Таким образом, блочные методы интегрирования обладают «внутренним» параллелизмом, поскольку допускают одновременное вычисление интегрируемых функций в нескольких точках временной оси моделирования, это хорошо согласуется с архитектурой параллельных вычислительных систем. Так, для методов с двухточечным и четырехточечным блоками обработка точек блока может осуществляться с использованием двух и четырех процессоров, соответственно.

Точность одношаговых параллельных блочных методов

В работе [2] показано, что для одношаговых блочных методов локальная ошибка в узлах блока имеет порядок $O(h^{k+2})$, где k – число узлов блока. В работе [3] разработана методика оценки точности параллельного моделирования движения КА вокруг собственного центра масс с помощью блочных одношаговых методов. Смысл методики состоит в сравнении результатов численного интегрирования уравнений движения КА с аналитическим решением, полученным для случая конического движения КА. На рис.1 приведен пример оценки погрешности моделирования движения КА различными методами интегрирования.

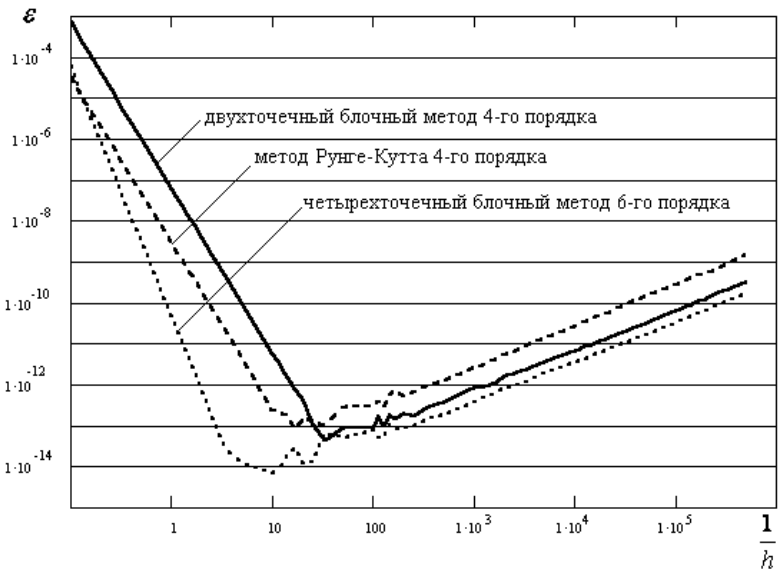


Рис. 1

Программа моделирования реализована на языке C с использованием функций библиотеки MPI реализации mpich 1.2.4. Формат представления данных – double. Моделирование выполнялось на параллельной системе, состоящей из рабочих станций Pentium IV (1,76 ГГц), объединенных сетью Fast Ethernet. Погрешность определялась по результатам интегрирования движения КА в течение 1000 секунд полета с различными шагами дискретизации. Для сравнения на рис.1 показаны результаты оценки погрешности моделирования для последовательного метода Рунге-Кутта 4-го порядка.

Исследование быстродействия параллельных блочных методов для решения задачи моделирования движения КА

Общепринято для оценки качества параллельных алгоритмов и программ использовать такие характеристики, как ускорение и эффективность:

$$S = \frac{T_1}{T_p},\ E = \frac{S}{P}, \tag{3}$$

где T_1 – время решения задачи на одном процессоре, T_p – время решения задачи на p процессорах.

Формулы для оценки T_1 и T_p в случае двухточечного метода можно получить из анализа (1) (детально алгоритмы параллельных блочных методов представлены в работе [4]):

$$T_1 = 17\, t_m\, n + 19\, t_s\, n + 7\, t_f, \tag{4}$$

$$T_p = 7\, t_m\, n + 10\, t_s\, n + t_c + 4\, t_f, \tag{5}$$

где n – размерность вектора неизвестных интегрируемой системы уравнений;

t_s, t_m – время, затрачиваемое на операции сложения и умножения, соответственно;

t_c – время, необходимое для обмена данными между процессорами;

t_f – время вычисления правых частей уравнений.

Очевидно, что ускорение зависит от сложности моделируемой системы и характеристик вычислительной системы. Как правило, основные затраты времени при численном интегрировании уравнений, описывающих движение КА, составляют затраты на вычисления правых частей системы уравнений, т.е. t_f в (4) и (5).

С целью экспериментального исследования эффективности применения параллельных вычислительных систем для моделирования задач данного класса в качестве меры вычислительной сложности решаемых уравнений нами взято время, необходимое для выполнения операции умножения вещественных чисел. Последовательно увеличивая сложность исходной задачи (оцененной как 10^4 операций умножения), мы получили зависимости ускорения от количества операций в правых частях дифференциальных уравнений, описывающих моделируемую систему. Измерение затрат времени проводилось для интегрирования 10000 секунд полета КА с шагом интегрирования одна секунда.

Эксперименты проводились на двух параллельных системах. В качестве первой параллельной системы использовалась компьютерная сеть типа Fast Ethernet с рабочими станциями Intel Pentium IV (1.76 ГГц). В качестве второй – SCI-кластер Петродворцового

телекоммуникационного центра[3]. SCI-кластер состоит из 20 двухпроцессорных узлов, тип процессоров – Intel Pentium III Copermine (933 МГц), объем ОЗУ на узле 512 Мб, способ соединения узлов – однонаправленные кольца SCI для передачи данных в системе MPI и сеть Fast Ethernet для управления машинами.

На SCI-кластере были проведены два варианта экспериментов, условно названные single и double. В варианте single на каждом узле кластера использовался только один процессор, т.е. для двухточечного метода интегрирования была задействована одна связь типа SCI, а для четырехточечного – три. В варианте double использовались оба процессора в узлах кластера, следовательно, двухточечный метод интегрирования выполнялся в режиме SMP на одном узле, а четырехточечный – с использованием двух узлов в режиме SMP и одной SCI-связью между узлами.

Результаты вычислительных экспериментов приведены в табл.1. Для ссылок на результаты экспериментов введены следующие обозначения: Net – результаты, полученные на первой параллельной системе, SCI-single и SCI-double – результаты, полученные на второй параллельной системе в вариантах single и double соответственно.

Таблица 1.

Результаты вычислительных экспериментов

Количество вычислительных операций ($\times 10^4$)	Время вычислений при применении блочного метода, секунды									
	двухточечный					четырехточечный				
	последовательный		параллельный			последовательный		параллельный		
	Net	SCI-single/ SCI-double	Net	SCI-single	SCI-double	Net	SCI-single/ SCI-double	Net	SCI-single	SCI-double
1	3,23	3,80	5,11	3,46	3,11	4,86	7,84	8,83	2,93	2,83
2	6,34	4,54	6,93	3,87	3,55	9,48	8,97	10,14	3,26	3,10

[3] Авторы выражают свою признательность доцентам кафедры вычислительной физики Санкт-Петербургского государственного университета Комолкину Андрею Владимировичу и Немнюгину Сергею Андреевичу за содействие в организации экспериментов на SCI-кластере Петродворцового телекоммуникационного центра.

2,5	8,02	4,93	7,92	4,11	3,77	11,82	9,54	10,80	3,43	3,29
3	9,48	5,30	8,69	4,32	3,98	14,13	10,09	11,50	3,59	3,44
5	15,62	6,81	12,25	5,20	4,85	23,46	12,36	14,18	4,24	4,16
10	31,16	10,55	21,15	7,42	7,03	46,70	17,72	20,88	5,90	5,83
26	80,98	22,62	49,59	14,47	13,92	121,37	35,82	42,06	11,22	11,06
51	158,27	41,44	93,65	25,54	24,75	237,40	64,05	75,35	19,55	19,56
101	313,90	79,10	181,71	47,62	46,35	471,18	120,59	141,76	36,15	36,26
151	469,18	116,80	269,64	69,21	68,06	704,40	177,04	208,23	52,79	52,87

Для измерения затрат на обмены между процессорами были применены тесты скорости обменов, разработанные в НИВЦ МГУ [5]. Общие затраты времени на выполнение обменов в двухточечном и четырехточечном методах для рассматриваемых вычислительных систем представлены в табл.2.

Таблица 2.

Затраты времени на обмен данными

Блочный метод	Время, секунды		
	Net	SCI-single	SCI-double
Двухточечный	2,113	0,208	0,043
Четырехточечный	8,192	1,027	0,926

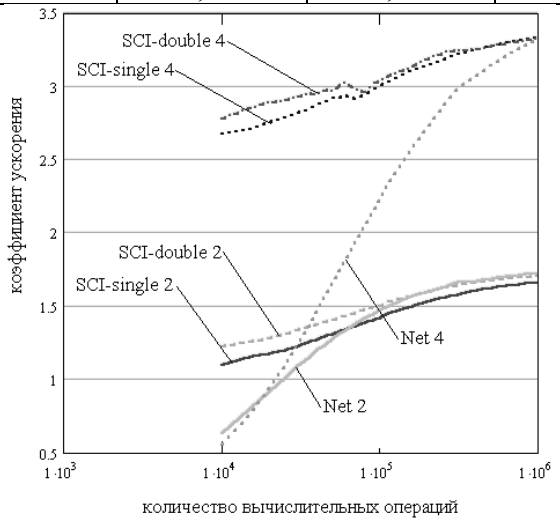


Рис. 2.

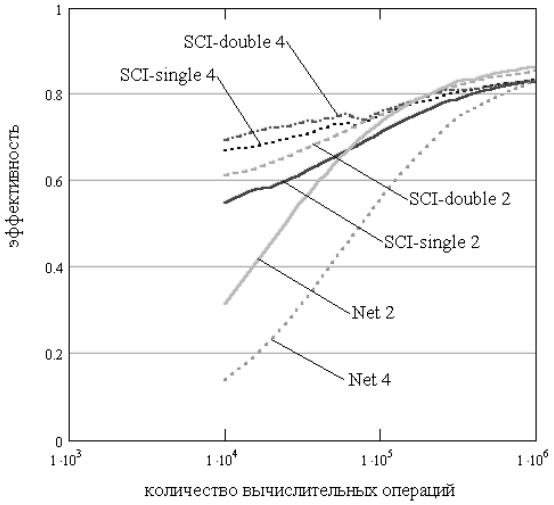


Рис. 3.

На рис. 2 и 3 показаны зависимости ускорения и эффективности параллельного решения задачи от принятой оценки вычислительной сложности правых частей системы уравнений. При этом используются введенные выше обозначения Net, SCI-single и SCI-double, а цифры 2 и 4 указывают, соответственно, на двухточечный и четырехточечный методы.

Полученные результаты можно интерпретировать следующим образом. Обозначим через m коэффициент в первой колонке табл.1. Тогда, используя оценки (4) и (5), общие затраты времени на вычисления на одном процессоре и параллельной системе для двухточечного метода интегрирования можно выразить как

$$T_1 = C_1 + 7m\,t_f\,, \tag{6}$$

$$T_p = C_p + t_c + 4m\,t_f\,, \tag{7}$$

где C_1 и C_p – суммарные затраты времени на последовательную и параллельную реализации метода интегрирования, соответственно.

Пользуясь результатами, приведенными в таблицах, можно определить C_1, C_p и t_f для каждой из реализаций двухточечного метода

интегрирования. Подставляя (6) и (7) в формулу для ускорения (3) и учитывая, что применение параллельной системы становится эффективным при $S > 1$, получим:

$$m > \frac{t_c + C_p - C_1}{3\,t_f}\,. \qquad (8)$$

Выводы

Таким образом, применение параллельных блочных методов становится эффективным при решении задачи моделирования движения КА на первой параллельной вычислительной системе, когда $m > 2.43$, т.е. оценка вычислительной сложности правых частей уравнений превышает $2.43 \cdot 10^4$. Для SCI–кластера значение m меньше нуля, это означает, что его применение эффективно уже для исходной постановки данной задачи, т.е. при оценке сложности меньше чем 10^4. Аналогичные условия можно получить и для четырехточечного метода.

Литература

1. Системы параллельной обработки / Под ред. Д. Ивенса. – М.: Мир, 1985. -416 с.
2. Фельдман Л.П., Дмитриева О.А. Разработка и обоснование параллельных блочных методов решения обыкновенных дифференциальных уравнений на SIMD-структурах // Наукові праці Донецького національного технічного університету. Вип. 29. – Донецьк, 2001. С. 70-79.
3. Пиза Н.Д., Кудерметов Р.К. Применение параллельных блочных методов для моделирования движения космического аппарата // Вісник технологічного університету Поділля. – 2003. – №3, Т.1. – С. 93-96.
4. Пиза Н.Д. Исследование эффективности применения параллельных вычислительных систем для моделирования движения космического аппарата // Радиоэлектроника. Информатика. Управление. – 2003. – №1. – С. 102-108.
5. Андреев А.Н., Воеводин В.В. Методика измерения основных характеристик программно-аппаратной среды www.dvo.ru/bbc/benchmarks.html.

ПРОСТРАНСТВЕННОЕ И ПОТОКОВОЕ РАСПАРАЛЛЕЛИВАНИЕ В ТЕХНОЛОГИЯХ ОБРАБОТКИ ИЗОБРАЖЕНИЙ СКОЛЬЗЯЩИМ ОКНОМ

С.Б. Попов, С.А. Скуратов

Институт систем обработки изображений РАН, г.Самара
Самарский государственный аэрокосмический университет

Двумерность можно считать основной особенностью изображений как данных, причем изображение является не просто матрицей чисел, а матрицей, соседние отсчеты которой определенным образом связаны между собой.

Несмотря на это, большинство алгоритмов и методов обработки изображений носят последовательный характер [1]. Это проявляется, с одной стороны, в структуре вычислений «внутри» алгоритма – очень популярно использование построчной, последовательной развертки отсчетов изображений и преобразований, имеющих локальный характер (поэлементных или на основе скользящего окна). С другой стороны, значительное число методов сложной обработки изображений могут быть реализованы как последовательное применение некоторых законченных типовых операций над изображениями [2]. Причем очень часто в качестве таких типовых операций выступают методы, использующие последовательную обработку отсчетов изображений «скользящим окном»:

$$g(m,n) = \Phi\left[\{f(m+k,n+l)\},(k,l) \in D\right],$$

где Ф – оператор преобразования отсчетов исходного изображения $\{f\}$; D – «окно» обработки, определенное относительно начала координат. Наиболее популярным видом окна обработки является квадратная окрестность, симметрично расположенная относительно текущего отсчета.

Рассмотрим некоторую типовую технологию обработки изображения, которая описывается следующим образом:

$$f_i(m,n) = \sum_{k=-K}^{K} \sum_{l=-L}^{L} a_i(k,l) f_{i-1}(m+k,n+l),$$

$$m = \overline{0,M-1}, n = \overline{0,N-1}, i = \overline{1,I}$$

т.е. к исходному изображению f_0 последовательно применяется I операций обработки. При $K = L = 1$ это соответствует локальной обработке скользящим окном 3×3, при $K = L = 2$ – окном 5×5 и т.д.

Очевидным вариантом распараллеливания вычислений в рамках данной технологии обработки изображений является декомпозиция по данным.

В случае наиболее популярной построчной организации хранения отсчетов изображения очень естественной представляется одномерная построчная декомпозиция, при которой каждая задача обрабатывает фрагмент изображения, содержащий смежные строки. Фрагменты перекрываются между собой на K строк, чтобы каждая задача на i-ом шаге имела все данные для обработки граничных точек. С этой же целью после каждого шага обработки задачи обмениваются перекрывающимися строками.

Время выполнения полной обработки одного изображения складывается из времени рассылки фрагментов по параллельным задачам, времени выполнения I шагов обработки, времени обмена перекрывающимися строками и времени формирования результирующего изображения из обработанных фрагментов.

Время рассылки фрагментов изображения определяется следующим образом:

$$T_0 = (P-2)\left(\frac{M}{P} + 2K\right)t_c(N) + \left(\frac{M}{P} + K\right)t_c(N),$$

где P – число параллельно исполняемых задач, $t_c(N)$ – время пересылки строки изображения длиной N точек. Время выполнения одного шага обработки составляет $T_{pi} = \frac{M}{P}t_p(N)$, где $t_p(N)$ – время обработки одной строки изображения. Время обмена перекрывающимися строками после выполнения одного шага обработки – $T_{ci} = 2Kt_c(N)$. Время формирования результирующего изображения из обработанных фрагментов – $T_{I+1} = (P-1)\frac{M}{P}t_c(N)$. Полное время обработки по данной технологии составляет $T_{1D} = T_0 + T_{I+1} + \sum_{i=1}^{I} T_{pi} + \sum_{i=1}^{I-1} T_{ci}$, т.к. обмен строками после последнего шага не производится, или окончательно

$$T_{1D} = 2\frac{P-1}{P} M\, t_c(N) + (2P-3) K\, t_c(N) + \frac{I\,M}{P} t_p(N) + 2K(I-1) t_c(N)$$

Относительная эффективность данного варианта параллельной обработки определяется по формуле

$$E_{1D} = \frac{I\,M\,t_p(N)}{P\,T_{1D}} = \frac{1}{1 + 2\dfrac{t_c(N)}{t_p(N)}\left(\dfrac{P-1}{I} + \dfrac{K\,P}{I\,M}(P + I - 2{,}5)\right)}.$$

Отношение $t_c(N) / t_p(N)$ тесно связано с вычислительной сложностью алгоритма обработки, чем проще алгоритм, тем больше это отношение, чем сложнее алгоритм, тем меньше это отношение.

Локальность обработки на каждом шаге обработки изображения позволяет применить альтернативную стратегию декомпозиции, основанную на функциональном подходе. Тот факт, что определенная часть информации готова значительно ранее полного завершения текущего шага обработки, позволяет использовать схему с параллельным выполнением всех шагов алгоритма и непосредственной передачей по мере готовности строк изображения от одной операции к другой.

В данном случае единицей разбиения служит операция, выполняющая один шаг обработки. Все операции выполняются параллельно. Данные передаются от одной операции к другой по мере готовности, т.е. после завершения обработки строки изображения на *i*-ом шаге, она передается другой задаче и является исходной для выполнения (*i*+1)-го шага. Фактически формируется поток данных изображения, элементы которого на *i*-ом процессоре подвергаются преобразованию в соответствии с алгоритмом *i*-ого шага и передаются последующему процессору для дальнейшей обработки. При этом результаты всего процесса обработки формируются с той же скоростью, с которой производится и ввод, но, после задержки пропорциональной числу выполняемых шагов обработки.

Время работы одной задачи складывается из времени обработки *M* строк изображения и пересылки их другой задаче $T_p = M\,t_p(N) + M\,t_c(N)$. Начало обработки *i*-ой задачи задерживается на $T_{idlei} = (i-1)(K+1)(t_p(N) + t_c(N))$ в случае $I \le P$, т.е. число шагов обработки не превышает число процессоров. Если число шагов

обработки превышает число процессоров, то задержка *i*-ой задачи составит

$$T_{idle\ i} = \left(i - \left[\frac{i-1}{P} \right] P - 1 \right) (K+1)\left(t_p(N) + t_c(N)\right) + \left[\frac{i-1}{P} \right] M\left(t_p(N) + t_c(N)\right),$$

где $[x]$ – обозначает целую часть числа x.

Общее время обработки по данному варианту параллельной обработки определяется временем завершения обработки на последнем шаге

$$T_{pipe} = T_{idle\ I} + T_p = \left(I - \left[\frac{I-1}{P} \right] P - 1 \right) (K+1)\left(t_p(N) + t_c(N)\right) +$$

$$+ \left(\left[\frac{I-1}{P} \right] + 1 \right) M\left(t_p(N) + t_c(N)\right)$$

Относительная эффективность потокового (конвейерного) варианта параллельной обработки определяется по формуле

$$E_{pipe} = \frac{I\,M\,t_p(N)}{P\,T_{pipe}} =$$

$$= \frac{1}{\left(1 + \frac{t_c(N)}{t_p(N)}\right)\left(\frac{P}{I\,M}\left(I - \left[\frac{I-1}{P}\right]P - 1\right)(K+1) + \frac{P}{I}\left(\left[\frac{I-1}{P}\right] + 1\right)\right)}.$$

Сравнение относительной эффективности параллельных реализаций рассматриваемой технологии обработки изображения приведено графиках. Сплошная линия описывает потоковый вариант, а пунктирная – вариант одномерной построчной декомпозиции.

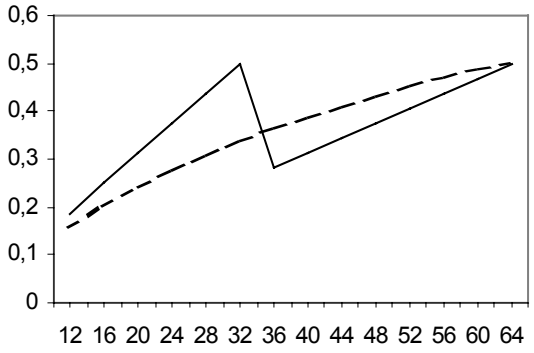


Рис. 1. Зависимость относительной эффективности от числа шагов обработки при $P = 32$, $M = 16384$, $t_c(N) / t_p(N) = 1$.

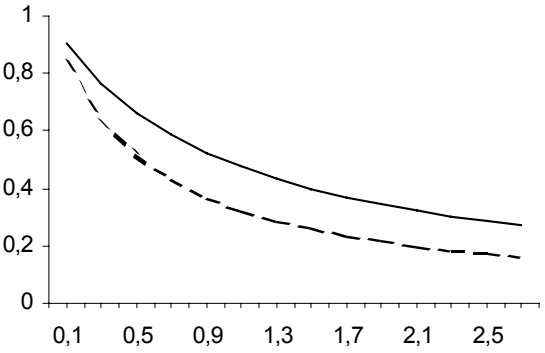


Рис. 2. Зависимость относительной эффективности от вычислительной сложности алгоритма обработки – $t_c(N) / t_p(N)$ при $P = 16$, $I = 16$, $M = 16384$.

Итак, для рассматриваемой технологии обработки изображений в случае, когда число шагов обработки близко, но не превышает число процессоров параллельной вычислительной системы, потоковый вариант реализации параллельной обработки изображений скользящим окном должен рассматриваться как достойная альтернатива

традиционному распараллеливанию с использованием декомпозиции по данным.

Работа выполнена при поддержке Министерства образования РФ, Администрации Самарской области и Американского фонда гражданских исследований и развития (CRDF Project SA-014-02) в рамках российско-американской программы «Фундаментальные исследования и высшее образование» (BRHE), а также при частичной поддержке РФФИ (грант № 01-01-00097).

Литература

1. Методы компьютерной обработки изображений / по ред. Сойфера В.А., М.: Физматлит, 2001, 780 с.
2. Попов С.Б., Сойфер В.А., Тараканов А.А., Фурсов В.А. Кластерная технология формирования и параллельной фильтрации больших изображений. // Компьютерная оптика, вып.23, Самара: ИСОИ РАН, 2002, с.75-78.

ПАРАЛЛЕЛЬНЫЕ СТРУКТУРЫ ПОТОКОВЫХ И ИТЕРАЦИОННЫХ АЛГОРИТМОВ РАСПРЕДЕЛЕНИЯ РЕСУРСА В ИЕРАРХИЧЕСКИХ СИСТЕМАХ

М.Х. Прилуцкий, Л.Г. Афраймович

Нижегородский государственный университет им. Н.И.Лобачевского

Общая математическая модель

Распределение ресурса осуществляется в многоуровневой иерархической системе, которая моделируется взвешенным ориентированным графом $G=(V,A)$, $A \subseteq V^2$, множество V вершин которого соответствует узлам системы, а множество A дуг – определяет связи между узлами. Пусть $(V_s,\ V_c,\ V_u)$ – разбиение множества V (соответственно, источники, коммуникационные узлы и потребители ресурса). Предполагается, что на элементах системы (на узлах и дугах) заданы сегментные ограничения, определяющие объемы ресурса, которые могут циркулировать в системе. Требуется определить такие допустимые объемы ресурса, которые соответствуют эффективной схеме функционирования системы.

Обозначим через $Q_v=\{i|(v,i)\in A\}$ – множество узлов системы, непосредственно следующих после узла v, $R_v=\{i|(i,v)\in A\}$ – множество узлов системы, непосредственно предшествующих узлу v, $v\in V$. Тогда $Q_j\neq\varnothing$, $R_j=\varnothing$ для $j\in V_s$; $Q_j\neq\varnothing$, $R_j\neq\varnothing$ для $j\in V_c$, $Q_j=\varnothing$, $R_j\neq\varnothing$ для $j\in V_u$.

Пусть $X=\left\|x_{ij}\right\|_{|V|\times|V|}$ матрица, элемент которой x_{ij} определяет количество ресурса, которое будет передано по дуге (i,j), $(i,j)\in A$. Ограничения математической модели включают в себя:

$$\sum_{j\in R_i} x_{ji} - \sum_{j\in Q_i} x_{ij} = 0\,,\ i\in V_k\,, \tag{1}$$

(коммуникационные узлы полностью передают весь поступивший к ним ресурс);

$$B_i \le \sum_{j\in Q_i} x_{ij} \le C_i\,,\ i\in V/V_u\,, \tag{2}$$

(источники ресурса способны производить, а коммуникационные узлы передавать, ограниченные объемы ресурса);

$$B_i \le \sum_{j\in R_i} x_{ji} \le C_i\,,\ i\in V_u, \tag{3}$$

(потребители ресурса получают ресурс в ограниченных объемах);

$$d_{ij} \le x_{ij} \le e_{ij}\,,\ (i,j)\in A, \tag{4}$$

(связи между узлами способны пропускать лишь ограниченные количества ресурса). Здесь $[B_i,C_i]$ – сегмент, соответствующий ресурсным ограничениям элемента i, $0\le B_i\le C_i<\infty$, $i\in V$; $[d_{ij},e_{ij}]$ – сегмент, соответствующий ресурсным ограничениям связи (i,j), $0\le d_{ij}\le e_{ij}<\infty$, $(i,j)\in A$.

В дальнейшем, для удобства изложения материала, ограничения (1) будем рассматривать в эквивалентном виде

$$\sum_{j\in R_i} x_{ji} - \sum_{j\in Q_i} x_{ij} \le 0\,,\ i\in V_k, \tag{5}$$

$$\sum_{j\in Q_i} x_{ij} - \sum_{j\in R_i} x_{ji} \le 0\,,\ i\in V_k. \tag{6}$$

Исследование общей математической модели

Специфика построенной общей математической модели (система линейных алгебраических неравенств транспортного типа) позволила для определения ее совместности разработать полиномиальный алгоритм. В его основу положена схема эквивалентного представления

многоуровневых иерархических систем в виде сетевых транспортных потоковых моделей с ограниченными пропускными способностями дуг. Использование для решения потоковых задач известных методов (например, модифицированный метод расстановки пометок для задачи о максимальном потоке [1], который требует $O(n^3)$ вычислительных операций) дало возможность построить эффективную процедуру проверки на совместность системы неравенств (2)–(6).

Постановки оптимизационных задач распределения ресурса

Среди множества узлов V выделим подмножество контролируемых K, которые определяют эффективность функционирования системы. Каждый из контролируемых узлов ν задает на соответствующем ему сегменте $[B_\nu, C_\nu]$ бинарное отношение, отражающее его предпочтение относительно соответствующего ему объёма ресурса, $\nu \in K$. Зададим бинарные отношения в виде функций предпочтения $\chi_\nu(x_\nu)$, определенных для контролируемых узлов, $x_\nu \in [B_\nu, C_\nu]$, $\nu \in K$. Задача распределения однородного ресурса в многоуровневой иерархической системе заключается в определение такого допустимого варианта распределения ресурсов, для которого функции предпочтения, определенные для контролируемых узлов, принимают экстремальные значения:

$$\chi_\nu(x_\nu) \to extr, \ \nu \in K.$$

Поставленная задача является задачей многокритериальной оптимизации, поэтому для ее решения необходимо выбрать схему компромисса. В работе [2] рассмотрена лексикографическая схема компромисса, в данной работе, предполагая, что распределение ресурсов в системе удовлетворяет условиям аддитивности и пропорциональности, будем рассматривать (как и в [3], но для общей структуры иерархической системы) линейные схемы компромисса.

Обозначим через d_ν, $d_\nu \geq 0$, – экономический показатель, соответствующий контролируемому узлу ν, $\nu \in K$. Тогда в качестве функций предпочтения выберем $\chi_\nu(x_\nu) = d_\nu x_\nu$, $\nu \in K$.

Пусть $D(X)=\left\{ X \middle| B_i \le \sum_{j\in Q_i} x_{ij} \le C_i, i \in V/V_u; \quad B_i \le \sum_{j\in R_i} x_{ji} \le C_i, \quad i \in V_u; \right.$

$\left. d_{ij} \le x_{ij} \le e_{ij}, (i,j) \in A; \sum_{j\in R_i} x_{ji} - \sum_{j\in Q_i} x_{ij} = 0, i \in V_c \right\}$ – множество

допустимых решений математической модели.

Задача минимизации затрат ($K=V\backslash V_u$):

$$F_1\left(X^0\right)= \min_{X\in D(X)} \sum_{i\in V/V_u} d_i \cdot \sum_{j\in Q_i} x_{ij} \ .$$

Задача максимизации дохода ($K=V_u$):

$$F_2\left(X^0\right)= \max_{X\in D(X)} \sum_{i\in V_u} d_i \cdot \sum_{j\in R_i} x_{ji} \ .$$

Задач максимизации прибыли ($K=V$):

$$F_3\left(X^0\right)= \max_{X\in D(X)} \left(\sum_{i\in V_u} d_i \cdot \sum_{j\in R_i} x_{ji} - \sum_{i\in V/V_u} d_i \cdot \sum_{j\in Q_i} x_{ij} \right).$$

Поставленные задачи являются задачами линейного программирования и для их решения могут быть использованы известные методы (например, симплекс-метод), однако специфика задач дала возможность применить для их решения более эффективные алгоритмы. Дугам канонической транспортной сети, моделирующей систему ограничений (2)–(6), кроме пропускных способностей, поставим в соответствие стоимости, определяемые экономическими показателями. Тогда решение каждой из поставленных задач заключается в поиске потока минимальной стоимости равного по величине максимальному потоку канонической транспортной сети. Для решения этих задач предлагается использовать прямой или двойственный алгоритмы поиска потока минимальной стоимости заданной величины [1], которые требуют $O(n^4)$ вычислительных операций. В случае большеразмерных систем решение поставленных задач требует значительных временных затрат. Для их решения здесь предлагается использовать модификацию метода ортогональных проекций Агмона-Моцкина [4], применительно к системе линейных алгебраических неравенств транспортного типа [5], позволяющий за счет «огрубления» результатов уменьшать время решения задачи. Для этого процедуру решения оптимизационной

задачи сведем к последовательной проверке на совместность системы линейных алгебраических неравенств, включающей неравенства транспортного типа, соответствующие ограничениям (2)–(6) и дополнительные линейные неравенства общего вида:

$$D^s \le F(X),\ F(X) \le G^s, \qquad (7)$$

здесь $[D^s, G^s]$ – сегмент возможных допустимых значений критерия решаемой задачи, s – номер шага алгоритма. Задав количество шагов алгоритма и изменяя границы сегмента возможных значений критерия, можно с заданной точностью найти решение поставленной задачи.

Параллельные структуры алгоритмов решения оптимизационных задач распределения ресурса

Для уменьшения времени решения большеразмерных задач распределения ресурсов в многоуровневых иерархических системах рассмотрим различные схемы распараллеливания процедуры решения задач на многопроцессорных системах.

Пусть h – количество параллельно работающих процессоров. При поиске максимального потока в сети, которая моделирует систему неравенств (2)–(6), предлагается в качестве начального потока использовать некоторый тупиковый поток. Для нахождения этого потока возможен одновременный поиск h различных прямых путей, увеличивающих поток. Тогда предлагается пропускать поток по одному из найденных путей, а затем, если какой-либо из найденных путей все еще увеличивает поток, то пропускать поток и через него, так просматривая все найденные пути. При поиске максимального потока методом расстановки пометок возможен одновременный поиск h различных путей, увеличивающих поток (обрабатывая их также как и прямые пути увеличивающие поток). При поиске максимального потока модифицированным методом расстановки пометок возможна параллельная обработка h точек одного слоя вспомогательной слоистой сети с целью ускорения операций «проталкивания» и «протягивания» потока (вспомогательных операций метода), а также ускорения процесса построения очередного слоя слоистой сети.

При решении задач, связанных с экономическими показателями, предлагается использовать обратный алгоритм поиска потока минимальной стоимости заданной величины, заключающийся в поиске цикла минимальной стоимости в вспомогательной сети. В данном

случае возможен параллельный поиск h различных циклов отрицательной стоимости. Тогда, изменив поток вдоль одного из найденных циклов, необходимо проверить оставшиеся циклы и если какой-нибудь из них все еще является отрицательным, изменить поток вдоль него. При использовании метода ортогональных проекций для проверки на совместность системы линейных неравенств (2)–(7) на очередном шаге предлагается разбивать сегмент $[D^s, G^s]$ на h+1 сегментов. Тогда параллельно проверив на совместность h систем линейных алгебраических неравенств, можно определить начальный сегмент для очередного шага. Метод Агмона-Моцкина предполагает последовательную проверку на совместность системы линейных неравенств до первого нарушения. Использование h-процессорной системы позволяет разбить все множество неравенств системы на h групп и обрабатывать каждую из групп параллельно.

Литература

1. Ху Т. Целочисленное программирование и потоки в сетях. М.: МИР, 1974. 519с.
2. Прилуцкий М.Х., Афраймович Л.Г. Параллельные алгоритмы распределения ресурсов в иерархических системах с лексикографическим упорядочиванием элементов. Материалы второго Международного научно-практического семинара «Высокопроизводительные параллельные вычисления на кластерных системах». Н.Новгород: Издательство Нижегородского госуниверситета, 2002. С 243-248.
3. Прилуцкий М.Х. Распределение однородного ресурса в иерархических системах древовидной структуры. // Труды международной конференции «Идентификация систем и задачи управления SICPRO’2000». Москва, 26-28 сентября 2000 г . М.: Институт проблем управления им.В.А.Трапезникова РАН, 2000. С.2038-2049.
4. Agmon S. The relaxation method for linear inequalities//Caned. J.Moth. 1954. V. 6. № 3. P. 382-392.
5. Прилуцкий М.Х. Многокритериальное распределение однородного ресурса в иерархических системах.// Автоматика и телемеханика. 1996. №2. С.24-29.

РАСПРЕДЕЛЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ РЕСУРСОВ НЕОДНОРОДНОЙ КЛАСТЕРНОЙ СИСТЕМЫ

М.Х. Прилуцкий, С.Ю. Петри

Нижегородский государственный университет им. Н.И.Лобачевского

Введение

Процесс управления решением совокупности задач в многопроцессорных системах формально можно представить как проблему распределения ограниченных ресурсов в сетевых канонических структурах (см. [1-2]), когда роль ресурсов играют вычислительные устройства и каналы связи, а процесс решения задачи, представленный в виде совокупности взаимозависимых деятельностей (подпрограмм, операций), моделируется ориентированным графом без петель и контуров, элементам которого поставлены в соответствие некоторые характеристики. Каноничность структуры означает, что никакая деятельность не может быть активизирована до тех пор, пока не завершится выполнение всех деятельностей ей предшествующих. Формальная постановка проблемы управления процессом решением совокупности задач в многопроцессорных системах в рамках математической модели распределения ресурсов в сетевых канонических структурах позволяет использовать хорошо разработанный аппарат решения задач распределения ресурсов к решению задач управления многопроцессорными системами.

Содержательная постановка задачи

Ресурсы, используемые в процессе выполнения деятельностей, будем подразделять на ресурсы общедоступного и ресурсы эксклюзивного типа. Общедоступный ресурс может одновременно потребляться несколькими деятельностями, а эксклюзивный ресурс может потребляться одновременно не более чем одной деятельностью. Например, при моделировании вычислительного процесса эксклюзивным ресурсом является канал передачи данных, а общедоступными ресурсами являются отдельные процессоры.

Как и в [1] будем предполагать, что процесс распределения ресурсов учитывает технологические, ресурсные, и организационные условия. К технологическим условиям относятся ограничения на

интенсивности потребления ресурсов (использование для выполнения подпрограмм нескольких процессоров), на последовательность и длительности выполнения деятельностей. К ресурсным условиям относятся ограничения на количества потребления деятельностями ресурсов. К организационным условиям относятся ограничения на возможные сроки начала и окончания выполнения деятельностей. Задача состоит в определение сроков начала и окончания выполнения деятельностей, интенсивностей расходования ресурсов таким образом, чтобы, не нарушая технологических, ресурсных и организационных условий, обеспечить выполнение всей заданной совокупности деятельностей.

Математическая модель

Обозначим через $T = \{1,..,T_0\}$- множество тактов планирования, J – множество различных деятельностей, I – множество ресурсов, используемых в системе. Будем предполагать, что ресурсы, используемые для активизации деятельностей, являются нескладируемыми (не использованные в каком-либо такте планирования, они не могут быть использованы в последующих тактах) и разбиваются на два подмножества: $I^{\text{Э}}$ – множество эксклюзивных ресурсов и $I^{\text{О}}$ – множество общедоступных ресурсов, $I^{\text{Э}} \cup I^{\text{О}} = I$, $I^{\text{Э}} \cap I^{\text{О}} = \varnothing$. Пусть V_{it} – количество ресурса i, которое поступит в систему в такт t, $i \in I$, $t \in T$. Обозначим через $K(j)$ – множество деятельностей, непосредственно предшествующих деятельности j, $K(j) \subset J$, $j \in J$. Через $R = (r_{ij})$ обозначим матрицу ресурсоемкостей, где r_{ij} обозначает количество ресурса i, которое требуется для выполнения деятельности j, $j \in J$, $i \in I$. Пусть m_{ij} и M_{ij}, соответственно, минимальная и максимальная интенсивности потребления (в один такт) деятельностью j ресурс i, $i \in I$, $j \in J$, $0 \le m_{ij} \le M_{ij} < \infty$; t_{ij}^{-} и t_{ij}^{+}, соответственно, минимальная и максимальная длительности потребления деятельностью j ресурс i, $0 \le t_{ij}^{-} \le t_{ij}^{+}$, $i \in I$, $j \in J$. Будем предполагать, что для некоторых деятельностей (из множества J^{H}) заданы h_j – начальные сроки – моменты времени (такты планирования), раньше которых деятельности j не могут начать выполняться (потреблять необходимые ресурсы); для некоторых деятельностей (из множества J^{D}) заданы d_j – директивные сроки –

моменты времени (такты планирования) завершения выполнения этих деятельностей.

Искомыми неизвестными математической модели являются:

- матрицы $X = (x_{ij})$ и $Y = (y_{ij})$, соответственно, времен начала и окончания активизации деятельностей, где x_{ij} – номер такта начала потребления деятельностью j ресурс i, y_{ij} – номер такта окончания потребления деятельностью j ресурс i, $x_{ij} \in T$, $y_{ij} \in T$;
- величины z_{ijt} интенсивности потребления деятельностью j ресурса i в такт t, $i \in I$, $j \in J$, $t \in T$;

Ограничения математической модели учитывают технологические, организационные и ресурсные условия.

Технологические условия:

$$\min_{i \in I} x_{ij} \geq \max_{l \in I} \max_{k \in K(j)} y_{lk}, \; j \in J \qquad (1)$$

(начало потребления деятельностью j ресурса i возможно не раньше, чем полностью будет выполнена любая деятельность, предшествующая деятельности j)

$$m_{ij} \leq z_{ijt} \leq M_{ij}, \text{ если } t \in \lfloor x_{ij}, y_{ij} \rfloor, \qquad (2)$$
$$z_{ijt} = 0, \qquad \text{если } t \notin \lfloor x_{ij}, y_{ij} \rfloor, \; i \in I, j \in J, t \in T.$$

(ограничения на интенсивности потребления ресурсов)

$$t_{ij}^{-} \leq y_{ij} - x_{ij} \leq t_{ij}^{+}, \; i \in I, j \in J, \qquad (3)$$

(ограничения на длительности выполнения деятельностей)

$$x_{ij} \geq y_{ik}, \text{ либо } x_{ik} \geq y_{ij}, \; i \in I^{\ni}, j \in J, k \in J. \qquad (4)$$

(условия на использование эксклюзивных ресурсов)

Организационные условия:

$$\min_{i \in I} x_{ij} \geq h_j, j \in J^H \qquad (5)$$

(выполнение заданных начальных сроков)

$$\max_{i \in I} y_{ij} \leq d_j, j \in J^D. \qquad (6)$$

(выполнение заданных директивных сроков)

Ресурсные условия:

$$\sum_{t \in T} z_{ijt} = r_{ij}, \; i \in I, j \in J, \qquad (7)$$

(для своего выполнения каждая деятельность должна полностью использовать необходимые ресурсы)

$$\sum_{j\in J} z_{ijt} \le V_{it},\ i\in I, t\in T. \tag{8}$$

(ресурсов должно быть достаточно для выполнения всех деятельностей)

$$x_{ij} \in \mathrm{T},\ y_{ij} \in \mathrm{T},\ z_{ijt} \ge 0,\ i\in I, j\in J, t\in T. \tag{9}$$

(естественные условия на введенные переменные).

Постановки задач

В рамках построенной общей математической модели ставятся различные оптимизационные задачи такие, как задача равномерного расходования ресурсов, задача наилучшего выполнения организационных условий по начальным и (или) директивным срокам и др. В качестве критериев оптимальности могут быть выбраны, например, следующие функционалы:

$$F_1 = \sum_{t\in T}\sum_{i\in I}\alpha_{it}\left(\frac{V_{it} - \sum_{j\in J} z_{ijt}}{V_{it}}\right)100 \to \min, \tag{10}$$

где α_{ij} – коэффициент штрафа за 1 процент не использования (потери) нескладируемого ресурса i в такт t, $i\in I$, $t\in T$.

$$F_2 = \sum_{j\in J^H} \beta_j (\min_{i\in I} x_{ij} - h_j) \to \min, \tag{11}$$

где β_j – коэффициент штрафа за 1 такт отклонения начала выполнения j- той деятельности от заданного начала, $j\in J^H$.

$$F_3 = \sum_{j\in J^D} \gamma_j (d_j - \max_{i\in I} y_{ij}) \to \min, \tag{12}$$

где γ_j – коэффициент штрафа, который получит система, если j-тая деятельность завершится раньше ее директивного срока на 1 такт, $j\in J^D$.

Фронтальный алгоритм решения задачи

В работе [2] показано, что для решения оптимизационных задач, которые могут быть поставлены в рамках общей математической модели с учетом всех ограничений (1)–(9), не существуют точные эффективные (отличные от полного перебора) алгоритмы. Отсюда,

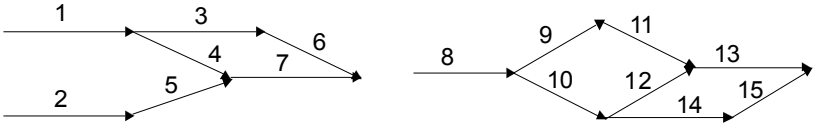
целесообразным является разработка и использование эвристических алгоритмов, применение которых позволит строить различные приближенные решения поставленных оптимизационных задач. В данной работе рассматривается фронтальный алгоритм. Этот алгоритм основан на идеологии жадных алгоритмов, то есть алгоритмов, в которых включенная в строящееся решение деятельность не может быть исключена из него на последующих шагах построения. Фронтом деятельностей называется множество $F(t_0)$, состоящее из всех деятельностей, каждая из которых может выполняться, начиная с момента времени t_0. Деятельность j включается в множество $F(t_0)$, если все ей предшествующие деятельности (из множества $K(j)$) завершили свое выполнение до момента времени t_0 и в системе присутствует необходимое количество ресурсов для назначения деятельности j на выполнение. На начальном этапе, используя ограничения (2) и (3), для каждой деятельности рассчитываются длительности t_{ij} – время потребления деятельностью j ресурса i, $j \in J$, $i \in I$. Исходя из величин t_{ij}, каждой деятельности присваивается длительность выполнения t_j, $t_j = \max\limits_{i \in I} t_{ij}$, $j \in J$. Таким образом, осуществляется переход от сетевой модели к сетевому графику, для которого можно провести расчет временных характеристик: найти моменты раннего начала выполнения деятельностей, раннего окончания выполнения деятельностей, позднего начала, позднего окончания выполнения деятельностей работ и резервы времени деятельностей. Затем множество $F(t_0)$ преобразуется в вектор $\overline{f}$, компонентами которого являются деятельности из множества $F(t_0)$, идущие в порядке не возрастания величин, соответствующим резервам времени деятельностей. Вектор $\overline{f}$ определяет порядок выбора деятельностей из фронта $F(t_0)$. Выбранная из фронта $F(t_0)$ очередная деятельность включается в строящееся расписание, причем для ее выполнения используются все доступные к данному моменту времени ресурсы в максимально возможном объеме. После назначения на выполнение, деятельность исключается из фронта и процедура формирования допустимого решения продолжается.

Распределение вычислительных ресурсов

Рассматривается многопроцессорная система, состоящая из процессоров разной производительности, соединенных в вычислительную сеть. На логическом уровне топология сети отображается полным графом. Будем считать, что пропускная способность каналов передачи данных одинаковая. В этой системе используется вычислительная модель передачи сообщений, реализованная при помощи MPI. Задания поступают в виде подпрограмм и операций, объединенных в программы. Последовательность подпрограмм и операций представляет собой каноническую сетевую структуру. Для подпрограмм известны требования на процессорное время, которое определяется производительностью процессора. Каждой подпрограмме выделяется отдельный процесс MPI. Требуется распределить процессы MPI по конкретным процессорам с учетом их производительности и загруженности. Передачу сообщений по каналам связи будем описывать операциями. Для каждой операции известно время, требуемое для передачи сообщения по соответствующему каналу связи.

Пример

Пусть на 4 процессорах требуется выполнить 2 задания.



Деятельности с номерами 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13, 14 являются подпрограммами, а деятельности с номерами 6, 12, 15 являются операциями. Перенумеруем ресурсы, используемые в рассматриваемом примере: пусть номера 1, 2, 3, 4 соответствуют процессорам, а номер 5 соответствует каналам связи. В таблице 1 приведены данные, связанные с необходимым обеспечением деятельностей ресурсами (в условных тактах – тактах планирования), с директивными сроками, штрафами и приоритетами деятельностей. В качестве критерия оптимальности выберем функционал

$$F_4 = \sum_{j \in J^D} \delta_j \max(0, (\max_{i \in I} y_{ij} - d_j)) \to \min ,$$

где δ_j – коэффициент штрафа, который получит система, если j-тая деятельность завершится после ее директивного срока на 1 такт, $j \in J^D$.

Исходные данные задачи отобразим в таблице 1.

Таблица 1.

Деятельности		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Ресурсы	1	10	12	7	6	5	-	18	10	4	15	12	-	17	14	-
	2	10	12	7	6	5	-	18	10	4	15	12	-	17	14	-
	3	13	15	10	9	8	-	21	13	7	18	15	-	20	17	-
	4	15	17	12	11	10	-	23	15	9	20	17	-	22	19	-
	5	-	-	-	-	-	10	-	-	-	-	-	5	-	-	4
Директивные сроки		-	-	-	-	-	30	37	-	-	-	-	-	47	39	43
Штрафы		-	-	-	-	-	4	5	-	-	-	-	-	3	6	5
Приоритеты		2	3	2	2	3	2	3	5	1	5	1	5	5	5	5

Шаги итераций приведены в таблице 2, причем рассматриваются только те такты планирования, в которых фронт работ не пуст.

Таблица 2.

Такт t	$V_{1,t}$	$V_{2,t}$	$V_{3,t}$	$V_{4,t}$	Фронт работ F(t)	Вектор $\overline{f}$	Назначенные деятельности
0	1	1	1	1	{1,2,8}	(8,2,1)	1,2,8
10	1	0	0	1	{9,10}	(10,9)	10,9
12	0	1	0	0	{5}	(5)	5
13	0	0	1	0	{3,4}	(3,4)	3
17	0	1	0	0	{4}	(4)	4
19	0	0	0	1	{11}	(11)	11
23	0	1	1	0	{6,7}	(7,6)	6,7
25	1	0	1	0	{12,14}	(12,14)	12,14
36	0	0	1	1	{13}	(13)	13
39	1	1	0	1	{15}	(15)	15

Все деятельности, назначенные на выполнение, завершились, фронт работ пуст, это означает останов алгоритма. Сроки начала и окончания выполнения деятельностей приведены в таблице 3.

Таблица 3.

Деятельности	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Ресурсы	3	2	3	2	2	5	2	1	4	1	4	5	3	1	5
Срок начала выполнения	0	0	13	17	12	23	23	0	10	10	19	25	36	25	39
Срок окончания выполнения	13	12	23	23	17	33	41	10	19	25	36	30	56	39	43
Директивные сроки	–	–	–	–	–	30	37	–	–	–	–	–	49	39	43

Таким образом, вся совокупность деятельностей выполнена за 49 условных тактов, причем деятельности 6, 7 и 13 завершены с нарушениями директивных сроков, соответственно, на 3, 4 и 7 тактов. Значение критерия на найденном решении $F_4 = 53$. Нижняя оценка значения критерия равна 47 и определяется из решения задачи при условиях, когда количество ресурсов у системы неограниченно.

Литература

1. Прилуцкий М.Х., Батищев Д.И., Гудман Э.Д., Норенков И.П. Метод декомпозиций для решения комбинаторных задач упорядочения и распределения ресурсов // Информационные технологии. 1997. N1. М., 1997. С.29-33/
2. Прилуцкий М.Х., Попов Д.В. Распределение и упорядочение работ в многостадийных системах // «Моделирование и оптимизация сложных систем». Межвузовский сборник научных трудов ВГАВТ. Н.Новгород, 1999. С. 84-93.

CMDE: ДИНАМИЧЕСКОЕ ПРОГРАММНОЕ ОКРУЖЕНИЕ ДЛЯ ОТКАЗОУСТОЙЧИВОГО ВЫПОЛНЕНИЯ MPI–ПРОГРАММ НА КЛАСТЕРАХ И СЕТЯХ

А.В. Селихов

Институт Вычислительной математики и математической геофизики СО РАН, г.Новосибирск

Введение

Одним из условий организации высокопроизводительных параллельных вычислений на кластерных системах является обеспечение отказоустойчивости выполняемых параллельных приложений к сбоям узлов этих систем. Это условие является необходимыи в случае, когда рассматриваются кластерные системы с большим количеством узлов и приложения, способные загрузить эти узлы на большой промежуток времени. Кластеры с несколькими тысячами узлов занимают в настоящее время верхние строки мирового рейтинга производительности, в то же время, среднее время между сбоями в таких системах составляет меньше одного дня.

Способы обеспечения отказоустойчивости приложений, выполняемых на кластерах и сетях, зависят, в первую очередь, от типа параллелизма, реализуемого задачей. В *распределённых* приложениях, подзадачи которых не взаимодействуют между собой во время выполнения, отказоустойчивость обеспечивается перезапуском отдельной подзадачи на другом узле системы с самого начала или с некоторой контрольной точки, благодаря чему приложение может быть завершено даже в случае постепенного уменьшения количества доступных узлов. Более сложным является обеспечение отказоустойчивости *параллельных* приложений, подзадачи которых взаимодействуют во время выполнения посредством разделяемых переменных или передачи сообщений. Далее рассматриваются параллельные приложения с передачей сообщений. В этом случае простой перезапуск подзадачи может привести либо к потере сообщений, отправленных подзадаче, но не полученных ей в результате сбоя, либо к посылке повторных сообщений вследствие повторного выполнения уже исполненного фрагмента программы, либо и того и другого вместе. Решение этой проблемы осуществляется различными способами, например, синхронизацией сохранения

процессами своих контрольных точек, или обеспечением отсутствия отправленных, но не принятых сообщений перед тем как сохранить контрольную точку. В обоих случаях целью является обеспечение непротиворечивости заний каждого параллельного процесса о состоянии всей системы в целом и отсутствие сообщений, находящихся в процессе передачи.

Система CMDE (Channel Memory based Dynamic Environment) [1] позволяет решать проблему восстановления выполнения параллельного приложения после сбоя его узла на основе использования Памяти Каналов (ПК), разделяющей процесс посылки и приёма сообщения между узлами и хранящей сообщения в промежутке между посылкой и приёмом, а также на основе сохранения всех сообщений, переданных через канал начиная с момента последней контрольной точки процесса-получателя сообщений. Основными достоинствами системы является возможность асинхронного создания контрольных точек параллельными процессами на каждом узле и независимое восстановление параллельных процессов после сбоя. Реализация этих возможностей потребовала введения в систему дополнительных компонент – Сервера Памяти Каналов (СПК) и Сервера Контрольных Точек (СКТ). Далее в работе представлены архитектура и принципы функционирования системы CMDE, а также некоторые результаты оценки её производительности.

Архитектура системы

Система CMDE является динамическим программным окружением, состоящим из множества взаимосвязанных компонент, каждый из которых запускается на отдельном узле кластерной системы или сети. Динамизм системы определяется обеспечением динамического включения компонентов в систему или исключения из неё.

Система CMDE состоит из двух подсистем: подсистемы низкоуровневых коммуникаций для библиотеки MPICH [2] и подсистемы запуска и сопровождения параллельных MPI-программ на кластерах и сетях. Первая подсистема состоит из одного компонента – коммуникационной библиотеки низкого уровня, реализующей всё множество высокоуровневых коммуникационных функций библиотеки MPICH. Вторая подсистема состоит из компонентов четырёх типов – Диспетчера, Сервера Памяти Каналов (СПК), Сервера

Контрольных Точек (СКТ) и Исполнителя. В текущей реализации системы для выполнения поставленных задач необходимо существование одного Диспетчера, одного или более СПК, одного или более СКТ и двух или более Исполнителей (рис.1.).

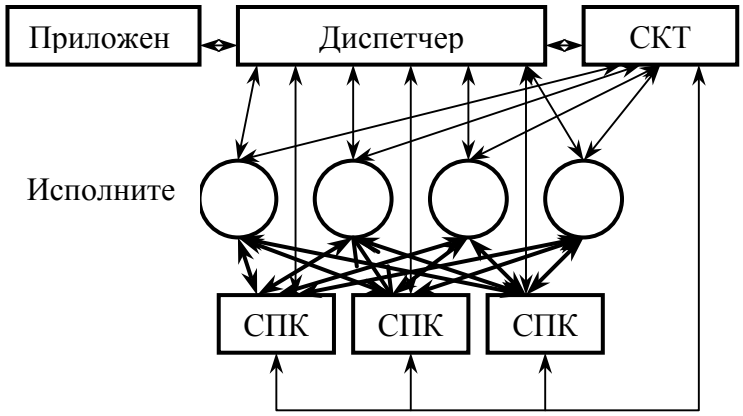


Рис.1. Пример структуры и связи в системе CMDE. Жирными стрелками выделены связи, непосредственно участвующие в выполнении приложения.

Диспетчер является центральным компонентом системы. Он обеспечивает добавление других компонентов в систему и отслеживает их отключение, формирует и поддерживает очередь приложений, назначает ресурсы приложению и перераспределяет ресурсы в случае сбоев.

Сервер Памяти Каналов (СПК) осуществляет буферизацию сообщений между процессами параллельного приложения и хранит все сообщения начиная с момента последней контрольной точки. Каждый СПК может обслуживать более одной очереди сообщений, представляющей память канала. При перезапуске процесса после сбоя СПК обеспечивает перезапущенный процесс сообщениями, полученными этим процессом в промежутке между последней контрольной точкой и сбоем.

Исполнитель является главным рабочим компонентом системы, обеспечивающим получение подзадачи параллельного приложения от Диспетчера, запуск подзадачи с начала или с последней контрольной

точки, инициирование генерации подзадачей контрольных точек на основе специального протокола [3] и отправка их на Сервер Контрольных Точек, а также контроль и анализ завершения подзадачи.

Сервер Контрольных Точек предназначен для получения, хранения и выдачи по запросу последних контрольных точек подпроцессов приложения. Контрольные точки различаются по уникальному идентификатору, определяемому в соответствии с номером приложения в системе и номером (ранком) подзадачи в приложении.

Коммуникационная библиотека низкого уровня обеспечивает подключение процесса подзадачи к Серверам Памяти Каналов для передачи сообщений на основе протокола TCP/IP, передачу служебных и пользовательских сообщений и генерацию контрольных точек на основе функций библиотеки Condor Stand-Alone Checkpointing Library (CSCL)[4] системы Condor[5]. Генерация контрольной точки осуществляется параллельно с помощью специально создаваемого процесса.

Функционирование системы

Функционирование системы CMDE можно условно разделить на две составляющие: формирование системы и обслуживание процесса запуска, выполнения, восстановления после сбоев и завершения параллельного приложения.

Формирование системы начинается с запуска Диспетчера на отдельном узле системы, при этом узел должен позволять входящие соединения с других узлов на специально выделенные порты. Диспетчер создаёт на диске базу приложений, регистрируемых в системе и ожидает соединений от других компонентов системы. Все остальные компоненты могут быть подключены в произвольном порядке, используя IP-адрес Диспетчера. Каждый новый компонент регистрируется Диспетчером и помещается во множество доступных ресурсов. Отключение компонентов от системы при отсутствии запущенных приложений приводит к исключению ресурсов из этого множества. Введение новых компонентов в систему возможно и во время выполнения параллельных приложений. Новые СКТ и Исполнители регистрируются как свободные ресурсы и могут быть использованы другим приложением, ожидающим достаточного количества ресурсов. Включение в систему нового СПК приводит либо к его включению в уже выполняющееся приложение, если

количество используемых этим приложением СПК меньше количества подзадач, либо к его регистрации в множестве свободных ресурсов.

Запуск приложения в систему требует подготовительного этапа, на котором исходный код(ы) приложения без модификации компилируется с использованием модифицированной библиотеки MPICH и библиотеки Condor. Конфигурация приложения, включающая в себя имя исполняемого файла, командную строку и имена всех входных файлов, описывается простым текстовым файлом, являющимся входным для специального Клиента, обеспечивающего необходимый интерфейс с Диспетчером для введения приложения в систему. На втором этапе с помощью Клиента все необходимые файлы и параметры пересылаются Диспетчеру, который ставит приложение в очередь или запускает его при наличии свободных ресурсов. В текущей реализации системы CMDE, СПК и СКТ являются ресурсами, разделяемыми между приложениями и наличие в системе двух СПК, одного СКТ и одного Исполнителя позволяет запустить приложение на выполнение. Исполнители являются выделенными ресурсами: исполнитель не может использоваться двумя подзадачами одновременно.

Обеспечение отказоустойчивости приложений

Использование системы CMDE на кластерах и сетях имеет целью обеспечение выполнения параллельных MPI-приложений на кластерах и сетях при наличии сбоев их компонентов. Сбой компонента (узла) кластера или сети считается фатальным, т.е. включение компонента в систему после сбоя осуществляется так же, как и его первое подключение. Реализация отказоустойчивости параллельного приложения в системе CMDE основывается на использовании Памяти Каналов, позволяющей хранить сообщения, переданные процессу параллельного приложения с момента последней контрольной точки до момента сбоя и восстанавливать эти сообщения при перезапуске процесса на другом узле. Контрольные точки процессов формируются с использованием библиотеки системы Condor. Использование в CMDE дополнительных компонентов требует обеспечения их отказоустойчивости или введения дополнительных требований на надёжность узлов кластеров или сетей. В настоящее время в системе существуют ограничения, связанные с надёжностью, только для двух её компонент: Диспетчера и Сервера Контрольных Точек. Серверы

Памяти Каналов, количество которых является критическим для производительности параллельных приложений, могут быть размещены на ненадёжных узлах, для которых допустимы сбои. Восстановление Сервера Памяти Каналов осуществляется на основе специально разработанного алгоритма дублирования Памяти Каналов между СПК, являющимися соседними на кольце из всех СПК. Такое дублирование влечёт за собой минимальные накладные расходы, связанные с копированием сообщений с одного сервера на другой, однако приводит к ограничению на частоту сбоев соседних СПК и имеет более сложный алгоритм восстановления. Этот же алгоритм используется для введения в приложение нового СПК, что позволяет уменьшить нагрузку на другие СПК.

Механизмы обеспечения отказоустойчивости, реализованные в системе, позволяют иметь следующие последствия для приложения при выходе из строя каждого из компонентов системы. Выход из строя узла с Исполнителем влечёт за собой перезапуск его подзадачи на другом узле при наличии свободных Исполнителей, или приостановки выполнения подзадачи до появления такого свободного ресурса. Выход из строя СПК приводит к приостановке коммуникаций всех подзадач, связанных с этим сервером и возобновления их работы после завершения процесса реконфигурации системы, или перезапуск всех подзадач приложения при невозможности реконфигурации. Выход из строя СКТ приводит к перезапуску всех подзадач связанного с ним приложения с использованием другого СКТ при их наличии, либо к приостановке приложения и его перезапуску при появлении нового СКТ. Выход из строя Диспетчера не отражается на работе запущенных приложений, однако после их завершения все компоненты системы завершают работу, а все задачи, находившиеся в очереди, должны будут быть вновь зарегистрированы с помощью клиента. В будущем возможна реализация восстановления очереди задач при перезапуске Диспетчера на том же узле.

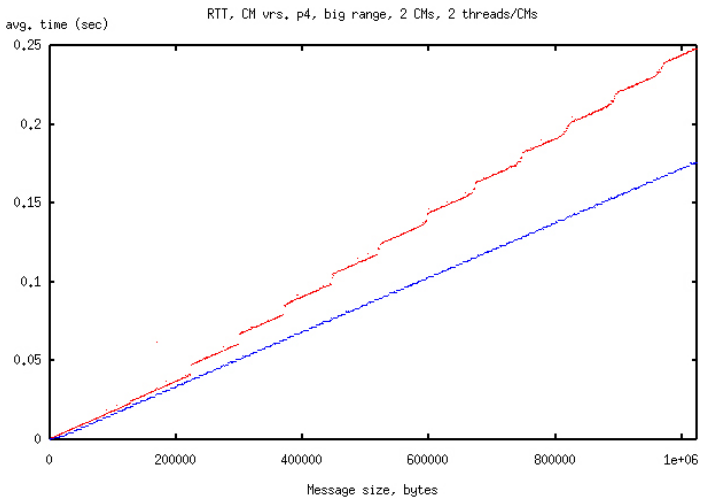


Рис. 2. Пропускная способность Серверов Памяти Каналов (верхняя кривая) по сравнению со стандартным MPICH

Производительность коммуникаций через Память Канала

Использование Памяти Канала в целях обеспечения отказоустойчивости параллельного приложения вводит дополнительные накладные расходы на время передачи сообщения от одной подзадачи к другой. Коммуникации «Подзадача 1 – Подзадача 2» распадаются на две составляющих: «Подзадача 1 – СПК» и «СПК – Подзадача 2». При синхронно взаимодействующих подзадачах время пересылки сообщения может удвоиться. Для уменьшения этих накладных расходов в СПК реализован транзитный конвейер, позволяющий отправлять сообщение одновременно с его получением. Результаты RTT(round-trip time) тестов пропускной способности Серверов Памяти Каналов в сравнении со стандартной реализацией MPICH (ch_p4) представлены на Рис.2. В этих экспериментах использовались два СПК, по одному на каждую подзадачу приложения. Такая конфигурация (одна подзадача на один СПК) является оптимальной с точки зрения пропускной способности СПК. При большем количестве подзадач, обслуживаемом одним СПК, его пропускная способность уменьшается. Результаты тестов приведены

для системы, не включающей протоколы обеспечения отказоустойчивости СПК.

Заключение

В работе представлена система CMDE, позволяющая запускать параллельные MPI-программы на кластерах или сетях, узлы которых не являются надёжными и могут быть выключены во время выполнения программы. Система поддерживает очередь приложений с помощью централизованного диспетчера и имеет Серверы Памяти Каналов и Серверы Контрольных точек для обеспечения восстановления выполнения работы параллельной программы после сбоя одного или нескольких её узлов. Все коммуникации между подзадачами приложений в системе осуществляются через Серверы Памяти Каналов, что позволяет сохранять и восстанавливать сообщения, принятые подзадачей в промежутке между последней контрольной точкой и сбоем, а также буферизировать и не передавать сообщения, отправляемые второй раз. Контрольные точки процессов подзадач создаются с использованием библиотеки системы Condor и сохраняются на специальном Сервере Контрольных Точек. В отличие от системы MPICH-V[6], в CMDE реализована отказоустойчивость Серверов Памяти Каналов, которая обеспечивается специальным алгоритмом дублирования сообщений. Система CMDE может быть использована для объединения вычислительных ресурсов различных кластеров и узлов сети с целью выполнения время- и ресурсоёмких MPI-программ.

Литература

1. Selikhov, A., Germain, C.: CMDE: a Channel Memory based Dynamic Environment for Fault-Tolerant Message Passing based on MPICH-V architecture. Proc. of 7-th Int. Conf. PaCT-2003, Russia, (2003) (to appear).
2. Selikhov, A., Bosilca, G., Germain, C., Fedak, G., Cappello, F.: MPICH-CM: A communication library design for a P2P MPI implementation. Proc. 9th European PVM/MPI User's Group Meeting, Linz, Austria, September/October 2002, LNCS, Vol.~2474. Springer-Verlag, Berlin Heidelberg (2002) 323–330.
3. Herault T., Lemarinier P.: A rollback-recovery protocol on peer to peer systems. In Proc. of MOVEP'2002 Summer School (2002) 313–319.

4. Condor Manuals, Chapter 4.2.1. http://www.cs.wisc.edu/condor/manual/
5. Raman R., Livny M.: High throughput resource management. Chapter 13 in The Grid: Blueprint for aNew Computing Infrastructure, Morgan Kaufmann, San Francisco, California (1999).
6. Bosilca G., Bouteiller A., Cappello F., Djilali S., Fedak G., Germain C., Herault T., Lemarinier P., Lodygensky O., Magniette F., Neri V., Selikhov A.: MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes. In Proc. IEEE/ACM SC2002 Conf., Baltimore, Maryland (2002).

МОДЕЛИРОВАНИЕ ДИНАМИКИ ПРОТОПЛАНЕТНОГО ДИСКА НА МНОГОПРОЦЕССОРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

А.В.Снытников

ИВМиМГ СО РАН, г. Новосибирск

Аннотация

Описывается вычислительная модель протопланетного диска на основе метода частиц в ячейках. Подробно расссмотрена методика распараллеливания, построенная с учетом физических особенностей задачи. Описаны вычислительные эксперименты, пространственное разрешение в которых соответствует требованиям решаемой задачи.

Введение

Эволюция самогравитирующих систем таких, как галактики или протопланетные диски имеет большой интерес для астрофизики Образование структуры диска представляет собой задачу многих тел в самосогласованном гравитационном поле. Хорошим приближением для моделирования самогравитирующего диска является кинетическое уравнение Власова-Лиувилля. Вместе с уравнением Пуассона для самосогласованного гравитационного поля они образуют систему уравнений звездной динамики [1]:

$$\begin{cases} \Delta\Phi = 4\pi G\rho \\ \dfrac{\partial f}{\partial t} + \vec{v}\nabla f - \nabla\Phi\dfrac{\partial f}{\partial \vec{v}} = 0 \end{cases}$$

Здесь диск галактики считается плоским и бесстолкновительным, и используется приближение дальнодействующего поля. Оценка ресурсов ЭВМ [2], требуемых для моделирования эволюции

протопланетного диска, дает сетку 1000x1000x500 узлов и порядка 100 млн. частиц. Для этого требуется 8 Gb оперативной памяти, таким образом, данная задача должна решаться на суперкомпьютерах.

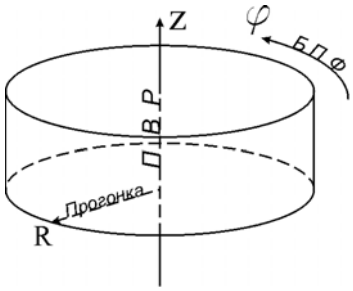
Описание численного алогоритма

Основную трудность в задаче моделирования эволюции протопланетного диска представляет расчет потенциала самосогласованного гравитационного поля, который определяется из уравнения Пуассона. Для того, чтобы получить быстрый алгоритм решения уравнения Пуассона, необходимо учесть особенности задачи и сделать алгоритм хорошо распараллеливаемым, то есть допускающим применение произвольного числа процессоров.

Рассмотрим особенности рассматриваемой задачи. Во-первых, она является нестационарной. Это означает, что физические величины – гравитационный потенциал, плотность и поле скоростей не могут изменяться скачкообразно. То есть, значение потенциала на соседних временных шагах не могут заметно отличаться. Следовательно, необходим метод решения, способный использовать значение потенциала, полученное на предыдущем временном шаге. Этому требованию соответствуют итерационные методы. Однако они являются недостаточно быстродействующими, поэтому предлагается комбинированный алгоритм, являющийся сочетанием прямых и итерационных методов.

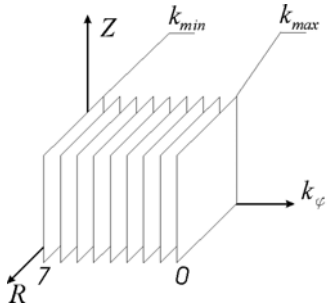
Во-вторых, протопланетный диск можно считать плоским, то есть толщина диска много меньше его радиуса, и плотность не равна 0 только на поверхности диска. Это означает, что изменение плотности вследствие сдвига частиц на некотором временном шаге приводит к заметному изменению потенциала только вблизи поверхности диска. Напротив, в узлах, удаленных от диска, эти изменения незначительны и условие сходимости выполняется гораздо быстрее. Таким образом, время счета значительно сокращается за счет того, что, удаленные от диска узлы не участвуют в расчете.

Уравнение Пуассона решается на сетке в цилиндрической системе координат для того, чтобы учесть симметрию диска и избежать появления нефизических структур, которые возникают при расчете в прямоугольных декартовых координатах.



Общая структура метода решения показана на рисунке. Первый шаг – быстрое преобразование Фурье по угловой кордината, в результате чего получается система линейных алгебраических уравнений на фурье-гармоники потенциала, каждое из которых независимо друг от друга. Далее двумерное уравнение для каждой гармоники решается методом верхней релаксации с прогонкой по радиальной координате. Общая структура метода показана на следующем рисунке. После окончания итерационного процесса выполняется обратное преобразование Фурье для значений потенциала в плоскости диска.

Распараллеливание



Одной из основных задач распараллеливания является минимизация обмена данными между процессорами. В предлагаемом методе решения уравнения Пуассона их удается полностью исключить за счет того, что уравнения для Фурье-гармоник потенциала не зависят друг от друга. Таким образом, появляется возможность разделить область решения на полностью независимые подобласти по угловым волновым числам, как показано на рисунке. Разделение области

равномерное, каждый процессор получает одинаковое количество гармоник.

Частицы разделяются между процессорами равномерно, без учета их расположения в пространстве. Это означает, что для расчета движения частиц каждый процессор должен знать распределение потенциала во всей плоскости диска.

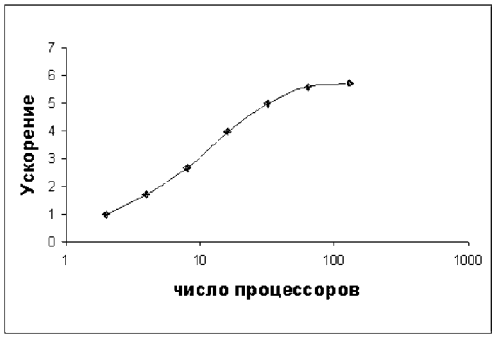
Межпроцессорные коммуникации в программе реализованы с помощью коллективных операций библиотеки MPI. На каждом временном шаге процесса моделирования обмен данными между процессорами выполняется дважды. Во-первых, после завершения итераций выполняется сборка гармоник потенциала в плоскости диска для обратного преобразования Фурье. Затем, после вычисления плотности в каждом процессоре их вклады суммируются.

Вычислительные эксперименты

Вычислительные эксперименты проводились на суперкомпьютере МВС-1000М в ССКЦ, г.Новосибирск и в МСЦ, г.Москва. Отладка программы проводилась на рабочей станции с двумя процессорами PentiumIII под управлением ОС Linux. Параметры некоторых расчетов приведены в следующей таблице:

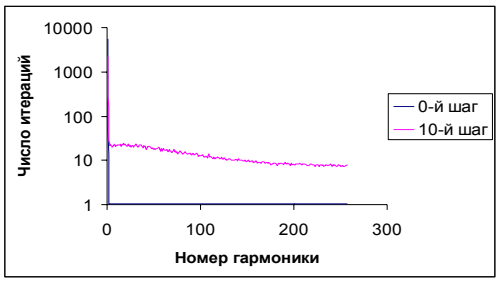
Число процессоров	Размер сетки	Число частиц	Время расчета одного временного шага, с
8	80 млн.	20 млн	7.2
64	2 млрд.	1 млрд.	141
64	3.2 млрд.	400 млн.	229.8
128	3.2 млрд.	400 млн.	180

На следующем графике приведено ускорение расчета на отладочной сетке 80 млн. узлов с 20 млн. частиц.



Таким образом, для небольшого числа процессоров эффективность распаралеливания составила 85%. При увеличении числа процессоров ускорение ухудшается и свыше 128 процессоров практически отсутствует. Причину этого можно прояснить с помощью следующего графика, на котором показана зависимость числа итераций, требуемое для достижения сходимости для каждой гармоники потенциала.

Распределение плотности изначально симметрично по угловой координате. Поэтому после пребразования Фурье только нулевая гармоника не равна 0, и сходимость для всех остальных гармоник достигается за 1 итерацию. Очевидно, на первом шаге работает только один процессор, содержащий эту гармонику. В дальнейшем картина изменяется (на графике показан 10-й временной шаг процесса моделирования), однако все равно расчет гармоник с малыми номерами занимает больше времени.



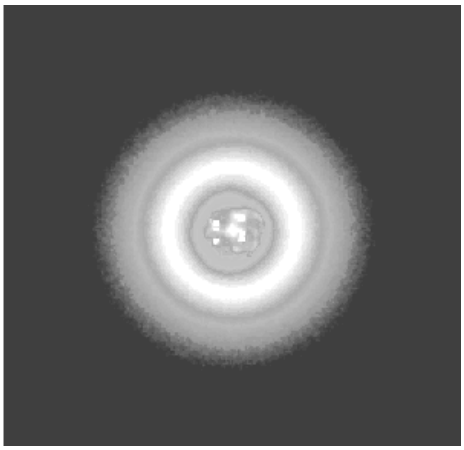
Таким образом, процессоры во время решения уравнения Пуассона оказываются неравномерно загруженными, причем неравномерность усиливается с увеличением числа процессоров: фактически, работает только тот процессор, который содержит нулевую гармонику. Эта проблема может быть решена, например, путем распараллеливания решения двумерного уравнения для нулевой гармоники потенциала, или использования для этой гармоники другого численного метода.

Важно отметить, что для различных сеток значения ускорения отличаются. При переходе с 64 процессоров на 128 на графике время расчета уменьшилось только на 2%. При увеличении сетки время расчета растет быстрее, чем время на коммуникации. Так, для сетки 3.2 млрд. узлов (см. таблицу) при переходе с 64 процессоров на 128 расчетное время уменьшилось на 10%.

В длительных расчетах проявляется особенность задачи, связанная с ее нестационарной природой. Проводилось моделирование развития неустойчивости в протопланетном диске. Использовалось 8 процессоров МВС-1000М в МСЦ, г.Москва, расчетная сетка в 256 млн. узлов. Продолжительность расчета 30 тыс. временных шагов. Полное время вычислений 25 суток (расчет проводился с сохранением данных и последующим продолжением счета), среднее время одного временного шага – 70 секунд.

Особенность состоит в следующем. В начальный период неустойчивости нет, существенных изменений в распределении гравитационного потенциала в течение одного временного шага не происходит, поэтому его значение для следующего временного шага вычисляется очень быстро. В дальнейшем, когда неустойчивость начинает развиваться, для вычисления потенциала итерационным методом требуется существенно больше времени. Это приводит к тому, что астрономическое время расчета одного временного шага в начале и в середине вычислительного эксперимента различается на порядок (12 и 100 секунд соответственно). Таким образом, действительное время расчета невозможно предсказать заранее.

Следующий рисунок показывает распределение плотности на диске, соотвествующее временному шагу 22500 (период затухания неустойчивости). Черный цвет соответствует нулевой плотности, шкала логарифмическая.


T=234

Выводы

Проведены расчеты, параметры которых соответствуют требованиям задачи о моделировании протопланетного диска. Для проведения длительных расчетов требуется дальнейшее распараллеливание программы До 32 процессоров эффективность распараллеливания равна 85%.

Литература

1. Хокни Р., Иствуд Дж. «Численное моделирование методом частиц», М. «Мир», 1987.
2. Вшивков В.А., Малышкин В.Э., Снытников А.В., Снытников В.Н. «Численное моделирование гравитационной динамики многих тел методом частиц в ячейках: параллельная реализация». СибЖВМ, том 6, номер 1, 2003. С. 25-36.
3. Краева М.А., Малышкин В.Э. «Динамическая балансировка загрузки в реализации PIC – метода на MIMD мультикомпьютерах», Программирование, № 1, 1999.
4.

РАСПРЕДЕЛЁННЫЕ ВЫЧИСЛЕНИЯ ПСЕВДОСИММЕТРИИ КРИСТАЛЛОВ

Н.В. Сомов, Е.В. Чупрунов

Нижегородский государственный университет им. Н.И.Лобачевского

Вычисление псевдосимметрии кристаллических структур базируется на расчёте функционала вида:

$$\eta_{\hat{g}}\left[\rho(\vec{r})\right] = \frac{\int_V \rho(\vec{r}) \cdot \rho(\hat{g} \cdot \vec{r}) dV}{\int_V \rho^2(\vec{r}) dV}, \qquad (1)$$

где $\hat{g}$ – операция симметрии, относительно которой рассчитывается степень инвариантности функции электронной плотности кристалла; V – объём элементарной ячейки; $\rho(r)$ – функция электронной плотности кристалла.

При вычислении данного функционала классическим способом используется свёртка двух функций электронной плотности в

обратном пространстве. Функция электронной плотности в обратном пространстве представляется в виде ряда Фурье, коэффициентами которого также являются ряды Фурье. Сложный набор данных и невозможность использования быстрого преобразования Фурье приводит к значительным вычислительным затратам. В данной работе нами предлагается новый способ расчёта данной величины.

Электронная плотность атома представляет собой положительную функцию, быстро убывающую до нуля и имеющую максимум в центре данного атома. Если предварительно рассчитать электронную плотность для каждого атома, то функцию электронной плотности кристалла можно приближённо представить как суперпозицию функций электронных плотностей атомов структуры.

Замена непрерывной функции электронной плотности кристаллической структуры дискретной позволит заменить интегрирование суммированием:

$$\eta_{\hat{g}}\left[\rho(\vec{r})\right] = \frac{\sum_{i=0}^{N_1}\sum_{j=0}^{N_2}\sum_{k=0}^{N_3}\rho_{ijk} * \rho_{\hat{g}(i,j,k)}}{\sum_{i=0}^{N_1}\sum_{j=0}^{N_2}\sum_{k=0}^{N_3}\rho^2{}_{ijk}} \qquad (2)$$

где ρ – массив электронной плотности, i, j, k – индексы массива электронной плотности, g – операция симметрии инвариантность относительно которой определяется. Операция симметрии g в данном случае действует на вектор индексов массива.

Распределение вычислений проводится следующим образом: Каждый процесс получает массивы электронной плотности и занимается обработкой отведённого участка. Нулевой процесс занимается формированием массива электронной плотности и рассылкой его процессам. Закончив вычисления, процессы передают результаты нулевому процессу, который в свою очередь, обрабатывает результаты работы процессов и формирует отчёт. Использование такого метода позволяет эффективно изменять точность расчётов и отказаться от вычисления рядов Фурье, что значительно может сократить время вычислений.

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ В ЗАДАЧАХ ВЫБОРА ГЛОБАЛЬНО-ОПТИМАЛЬНЫХ РЕШЕНИЙ ДЛЯ МНОГОПРОЦЕССОРНЫХ КЛАСТЕРНЫХ СИСТЕМ[4]

Р.Г. Стронгин, В.П. Гергель

Нижегородский государственный университет им. Н.И.Лобачевского

В работе рассматривается новой схемы параллельных вычислений, позволяющей существенно ускорить процесс поиска глобально-оптимальных решений по сравнению с лучшими последовательными методами при использовании многопроцессорных компьютерных систем кластерного типа. Предлагаемая схема характеризуется высокими показателями масштабируемости и надежности и является однородной, т.е. не требует наличия какого-либо единого управляющего центра для организации параллельных вычислений. В работе содержится описание модели вычислений для одного класса вычислительно-трудоемких задач, излагается схема эффективного распараллеливания вычислений для многопроцессорных кластерных систем и дается общая характеристика эффективных способов представления поисковой информации в процессе параллельных вычислений.

Модели вычислений для одного класса вычислительно-трудоемких задач

1. Исследование многих математических моделей, рождаемых потребностями приложений, часто включает построение оценок некоторой величины, характеризующей заданную область Q в многомерном евклидовом пространстве R^N. В качестве типового примера таких задач может рассматриваться, например, *задача нелинейного программирования*

$$\varphi^* = \varphi(y^*) = \min\{\varphi(y) : y \in Q\}, \quad (1)$$

где *область поиска*

$$Q = \{y \in D : g_i(y) \le 0, 1 \le i \le m\} \quad (2)$$

определяется в некотором N –мерный гиперпараллелепипеде

[4] Данная работа выполнена при поддержке Минобразования РФ (грант № Е02-1.0-58).

$$D = \left\{ y \in R^N : a_j \le y_j \le b_j, 1 \le j \le N \right\} \quad (3)$$

системой нелинейных ограничений $g_i(y) \le 0, \ 1 \le i \le m$. В случае многоэкстремальности функции $\varphi(y)$ рассмотренная постановка называется *задачей глобальной оптимизации*, для которой искомые оценки $(y^*, \varphi^* = \varphi(y^*))$ являются *интегральными характеристиками* минимизируемой функции $\varphi(y)$ во всей области D.

Рассмотренный пример может быть расширен аналогичным описанием задач интегрирования, решения систем нелинейных уравнений, восстановления зависимостей и т.п. Как результат такого перечисления можно говорить о **существовании широкого класса важных прикладных задач**, требующих оценки некоторой величины (интеграла, глобального минимума, множества не доминируемых решений и т.п.) путем анализа поведения заданной функции

$$F(y) = \left(F_1(y), \ldots, F_s(y)\right) \quad (4)$$

в некотором гиперпараллелепипеде D (в общем случае, функция $F(y)$ может быть векторной для учета, например, многокритериальности или ограничений задачи оптимизации).

Далее для конкретности проводимых рассуждений весь последующий материал данной работы излагается на примере задач глобальной (многоэкстремальной) оптимизации.

2. Задача построения искомой оценки могла бы быть решена на основе анализа вычисленных в узлах некоторой сетки

$$Y_T = \left\{ y^t : 1 \le t \le T \right\} \quad (5)$$

области D множества значений

$$Z^t = F(y^t), \quad y^t \in D, \ 1 \le t \le T \ . \quad (6)$$

Возможный (и широко используемый в практических приложениях) способ построения таких сеток состоит в равномерном расположении узлов (равномерность расположения узлов позволяет уменьшить их общее число T и, следовательно, сократить вычислительные затраты). Однако проблема состоит в том, что число узлов этой сети экспоненциально увеличивается с ростом размерности N области D. Действительно, для равномерной сетки в области D, имеющей шаг $\Delta > 0$, справедлива оценка

$$T = T(\Delta) \approx \prod_{j=1}^{N} \left[(b_j - a_j) / \Delta \right].$$

В результате для многих актуальных прикладных задач **реализация описанной схемы полного перебора на равномерной сетке оказывается невозможной** в силу ее высокой потребности в вычислительных ресурсах.

3. Дело усложняется еще и тем, что во многих приложениях выполнение операции оценки вектора $F(y)$ для выбранной точки $y \in D$ связано с вычислительным анализом математических моделей, отличающихся высокой степенью сложности. Причем практика последних десятилетий показывает, что впечатляющий воображение быстрый рост производительности компьютеров сопровождается столь же быстрым усложнением рассматриваемых прикладных задач и описывающих их моделей. Разумеется, существуют и многие более простые задачи, решение которых вполне осуществимо описанным выше методом перебора узлов равномерной сетки. Основное предложение, направленное на повышение эффективности анализа сложных задач, связано с переходом к целенаправленному анализу вариантов, в ходе которого вычисления, осуществленные в одних узлах сетки, позволяют исключить необходимость вычислений во многих других узлах. Т.е. речь идет об использовании существенно не равномерных сеток. Так, например, применительно к рассмотренной выше задаче глобальной минимизации это означает, что сетка должна быть более плотной в окрестности искомого глобального минимума и – заметно менее плотной вдали от точки минимума. По существу это означает, что такая сетка не может быть задана априори (ибо расположение глобального минимума является не известным). Ее необходимо строить в процессе анализа.

Основная идея сокращения объема вычислений, необходимых для оценки характеристик многомерной области, **состоит в использовании неравномерных сеток**, способных обеспечивать ту же точность решения, что и при равномерных сетках. При этом несколько узлов $y^1,\ldots, y^t$, $t \geq 1$ сетки Y_t могут быть заданы априори, однако, все остальные узлы y^k, $k \geq t$, последовательно определяются некоторым решающим правилом

$$y^{k+1} = G_k\left(y^1,\ldots,y^k; Z^1,\ldots,Z^k\right),\ k \geq t, \qquad (7)$$

где величины Z^l, $1 \leq l \leq k$, являющееся аргументами решающей функции G_k, есть значения функции $F(y)$, вычисленные в узлах y^l, $1 \leq l \leq k$ Каждая конкретная система решающих функций,

предлагаемая для определенного класса задач, **основывается на некоторых априорных предположениях о вектор–функции** *F*(*y*), что и позволяет использовать накапливаемую в процессе решения задачи информацию

$$\omega_k = \left\{ \; y^1, \ldots, y^k ; Z^1, \ldots, Z^k \; \right\} \qquad (8)$$

для построения неравномерных сеток.

Для наглядного представления рассмотренных положений приведем несколько иллюстративных примеров решения задач многоэкстремальной оптимизации.

Пример 1. На рис. 1 показана одномерная функция φ(*y*), $y \in [a, b]$, представляющая невязку аппроксимации для некоторой задачи восстановления зависимостей из работы Strongin and Sergeyev (2000). Функция невязки в этой задаче содержит 5 параметров. Однако 4 из них входят в выражение для φ линейно. Поэтому, полагая, что значение пятого параметра, роль которого играет указанная выше переменная *y*, является заданным, значения первых четырех параметров определяются путем решения соответствующей системы линейных уравнений. График функции, изображенный на рисунке, построен путем вычисления значений φ в 3750 узлах равномерной сетки, погруженной в область определения [0.5,8.0]. При этом была получена оценка $\tilde{\varphi} = 0.017$, достигающаяся в узле $\tilde{y} = 1.105$.

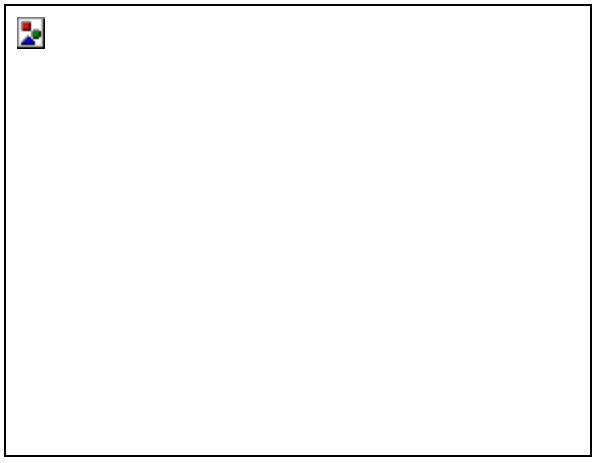


Рис. 1. Пример минимизации многоэкстремальной функции при использовании методов, порождающих разные неравномерные сетки в области поиска

Для сравнения эта же задача минимизации одномерной функции $\varphi(y), y \in [a, b]$ была решена тремя методами, порождающими неравномерные сетки. *Первый метод*, предложенный Пиявским (см. Пиявский (1972)), основан на предположении, что минимизируемая функция удовлетворяет условию Липшица. Верхний ряд штрихов, расположенных под осью абсцисс на рис. 1, указывает 95 точек неравномерной сетки (группы близких узлов обозначены темными прямоугольниками), порожденной алгоритмом Пиявского (называемым также методом ломаных). Множество узлов этой неравномерной сетки, обеспечивает ту же самую оценку, что и метод полного перебора на равномерной сетке, содержавшей 3750 узлов.

Второй метод, предложенный Кушнером (см. Kushner (1964)) и основанный на рассмотрении минимизируемой функции как некоторой реализации винеровского случайного процесса, также обеспечил оценки $\widetilde{\varphi} = 0.017$, $\widetilde{y} = 1.105$ за 95 итераций. Расположение точек этих итераций указывают штрихи из среднего ряда на рис.1.

Третий алгоритм (см. Стронгин (1978)) также основан на рассмотрении минимизируемой функции как реализации некоторого случайного процесса. Подобно двум предшествующим, этот алгоритм обеспечил ту же точность, что и метод перебора на равномерной сетке (остановка была произведена после выполнения 95 итераций). Точки итераций отмечены на рис. 1 штрихами нижнего ряда.

Сравнение трех рядов штрихов наглядно демонстрируют, что использование различных априорных предположений о поведении минимизируемой функции рождает разное расположение точек итераций. Однако во всех трех случаях достигается эффект уплотнения сетки в окрестности глобального оптимума. Важно обратить внимание на то, что **использование неравномерных сеток сократило необходимое число узлов почти в 40 раз.** Вместе с тем, следует иметь в виду, что выбор этих узлов требует запоминания и обработки (в соответствии с правилами (7)) информационного массива (8). Причем эти вычисления предшествуют каждой итерации. Однако время, необходимое для выбора очередного узла сетки обычно много

меньше времени оценки вектор–функции (4) в сложных прикладных задачах.

Пример 2. Рассмотрим класс двухмерных тестовых функций (см. Grishagin (1978))

$$\varphi(y) = \left\{ \left(\sum_{i=1}^{7} \sum_{j=1}^{7} \left[A_{ij} a_{ij}(y) + B_{ij} b_{ij}(y) \right] \right)^2 + \left(\sum_{i=1}^{7} \sum_{j=1}^{7} \left[C_{ij} a_{ij}(y) + D_{ij} b_{ij}(y) \right] \right)^2 \right\}^{1/2} \tag{9}$$

где

$$a_{ij}(y) = \sin(i\pi y_1)\sin(j\pi y_2), \quad b_{ij}(y) = \cos(i\pi y_1)\cos(j\pi y_2), \quad 0 \le y_1, y_2 \le 1,$$

и значения коэффициентов A_{ij}, B_{ij}, C_{ij}, D_{ij} выбираются как реализации случайной величины, равномерно распределенной на отрезке [–1,1]. Необходимость максимизации функций подобного вида возникает, например, в задачах оценки максимального напряжения (определяющего прочность) в упругих тонких пластинах при поперечной нагрузке. Это сходство с прикладными задачами привело к достаточно широкому использованию функций этого класса для оценки разрабатываемых алгоритмов глобальной оптимизации. Линии уровня двух конкретных функций из этого класса представлены на рис. 2.

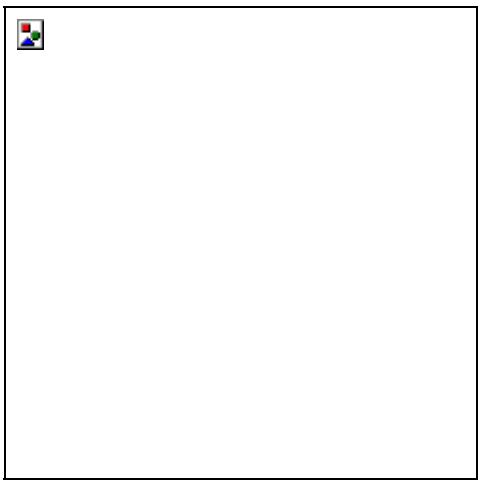


Рис. 2. Линии уровня и результаты оптимизации тестовой функции из семейства (9)

Минимизация функции, взятой с обратным знаком (в целях достижения глобального максимума), осуществлялась многомерным алгоритмом глобального поиска, использующим развертки типа кривой Пеано для редукции размерности задачи (см., Стронгин (1978, 1990)). Точки квадрата D, в которых выполнялись вычисления значений минимизируемых функций (124 итерации) отмечены на рисунке темными прямоугольниками. При этом фактическая точность решения задачи превышает точность метода полного перебора на сетке с шагом 0.01 (по каждой координате), использование которой потребовало бы 10^4 итераций. Т.е. **использование неравномерной сетки ускоряет вычисления (в рассматриваемом примере) почти на два порядка.** Для более полного представления о возможностях указанного алгоритма, была проведена оптимизация 100 функций из рассматриваемого семейства (9), результаты которой представлены оценкой операционной характеристики метода, график которой изображен на рис.3.

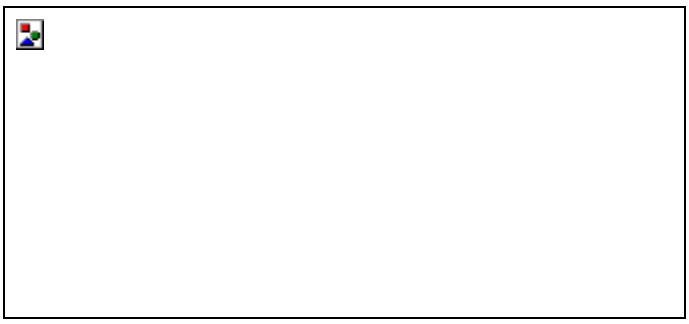


Рис. 3. Операционная характеристика многомерного алгоритма глобального поиска, оцененная на выборке из 100 функций семейства (9)

Представленная операционная характеристика показывает долю $P(k)$ функций из выборки, оценка глобального оптимума которых была найдена с точностью не хуже 0.01 по каждой координате за k итераций. Величина $P(k)$ нанесена на оси ординат, а расход ресурса (число итераций k) – на оси абсцисс. Как следует из рисунка, удовлетворительная оценка оптимума для более 80% примеров потребовала 200 лишь итераций. Ни одна задача не потребовала более 450 итераций.

Пример 3. Теперь рассмотрим пример двухмерной задачи (1) при $m = 3$, т.е. когда область $Q \neq D$. Минимизируемая функция и ограничения описываются выражениями:

$$\varphi(y) = -1.5x^2 \exp(1 - x^2 - 20.25(y_1 - y_2)^2) -$$
$$-[0.5(y_1 - 1)(y_2 - 1)]^4 \exp\{2 - [0.5(y_1 - 1)]^4 - (y_2 - 1)^4\},$$
$$g_1(y) = 0.01\left[(y_1 - 2.2)^2 + (y_2 - 1.2)^2 - 2.25\right] \le 0,$$
$$g_2(y) = 100\left[1 - (y_1 - 2)^2 / (1.2)^2 - (0.5y_2)^2\right] \le 0,$$
$$g_3(y) = 10\left[y_2 - 1.5 - 1.5\sin(6.283(y_1 - 1.75))\right] \le 0,$$

определенными в квадрате $0 \le y_1 \le 4$, $-1 \le y_2 \le 3$. Линии уровня функции и границы ограничений нанесены на рис. 4. Как видно из рисунка, допустимая область Q является не односвязной. Она состоит из трех частей, лежащих внутри окружности (представляющей граничную линию для первого ограничения), над линией эллипса

(представляющей множество принадлежащих квадрату D граничных точек второго ограничения) и под кривой синуса, соответствующей границе третьего ограничения. Точки 141 итераций, порожденных при решении этой задачи индексным алгоритмом глобального поиска (см., Стронгин, Маркин (1986)), отмечены на рисунке темными прямоугольниками. Точность полученной оценки оптимума составила 0.01 по каждой координате, что (как и в предшествующих примерах) существенно меньше числа узлов равномерной сетки с тем же шагом 0.01.

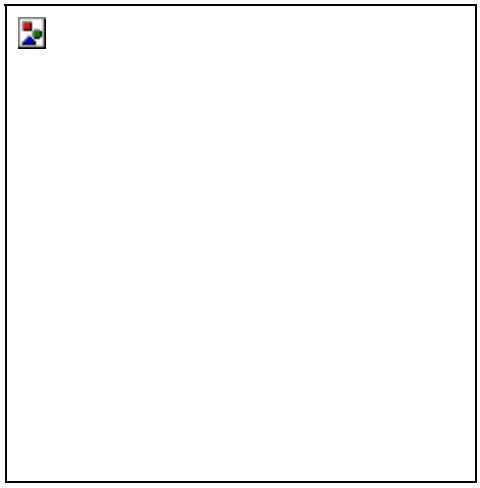


Рис 4. Линии уровня функции, линии границ допустимой области поиска и результаты глобального поиска

Эффективное распараллеливание вычислений для многопроцессорных кластерных систем

1. Многопроцессорные системы кластерного типа (см., например, Pfister (1995)) позволяют ускорить решение задач рассмотренного типа. Для задач, в которых определение значений функции $F(y)$ из (4)основано на требующем значительных компьютерных ресурсов численном анализе сложных математических моделей, наиболее целесообразным подходом для распараллеливания вычислений является одновременное вычисление значений функции $F(y)$ в нескольких узлах используемой сетки Y_t. При этом каждому из имеющихся $p > 1$ процессоров задается конкретная точка $y^l \in D$,

$1 \le l \le p$, в которой этот процессор (будучи загружен необходимым комплектом программ) определяет значение всех или части координатных функций $F_i(y^l)$, $1 \le i \le s$. Для реализации такого подхода решающее правило алгоритма должно определять одновременно $p > 1$ точек следующих итераций

$$(y^{k+1},\ldots,y^{k+p}) = G_{k,p}(\omega_k), \quad k \ge t, \quad y^{k+l} \in D, 1 \le l \le p, \qquad (10)$$

передаваемых отдельным процессорам (алгоритмы подобного типа, обобщающие эффективные последовательные схемы и характеризуемые низкой избыточностью, изложены в работе Стронгина и Сергеева (1989)).

2. Время выполнения испытаний в различных точках области поиска, в общем случае, является неодинаковым. Различное время вычисления координатных функций может проявиться, например, в результате применения экономичной схемы учета ограничений, когда нарушение любого из неравенств (2) интерпретируется как признак не допустимости анализируемого варианта, т.е. $y \notin Q$. В таком случае итерация в точке y может осуществляться путем последовательного определения значений координатных функций $F_i(y)$. При этом вычисления в точке y прекращаются после первого обнаружения нарушенного неравенства из (2). В связи с этим результаты вычислений в узле $y^l \in D$ могут быть представлены триадой

$$\omega^l = (\ y^l, \quad Z^l = (\ F_1(y^l),\ldots,F_\nu(y^l)\ ), \quad \nu = \nu(y^l)\ ), \quad 1 \le \nu \le s, \qquad (11)$$

где целое число $\nu = \nu(y^l)$ называется индексом точки y^l (вычисление триады (11) в узле y^l условимся называть испытанием в точке y^l). Именно такие частичные вычисления значений координатных функций осуществляются, например, при использовании индексных алгоритмов, предложенных в Стронгин и Маркин (1986), Стронгин (1990), Strongin and Sergeyev (2000) для решения задач вида (1)–(3).

С другой стороны, рассмотренная схема распараллеливания (10) определяет точки следующих p испытаний после получения результатов *всех* предшествующих испытаний, и, как результат, часть процессоров будет простаивать, ожидая завершения работы других процессоров. Этот недостаток может быть преодолен введением следующей *асинхронной схемы*.

Введем обозначение $y^{k+1}(j)$ для точки, в которой процессор с номером j, $1 \le j \le p$ осуществляет (k+1)-е испытание. При этом $k = k(j)$,

$1 \le j \le p$. В целях обеспечения компактности последующих записей, введем также единую нумерацию всех точек гиперинтервала D, в которых испытания уже завершены. Используя верхние индексы, определим множество

$$Y_\theta = \left\{ y^1, \ldots, y^\theta \right\} = \bigcup_{j=1}^{p} \bigcup_{i=1}^{k(j)} y^k(j), \tag{12}$$

в точках которого вычислены значения функции $F(y)$, составляющие массив результатов

$$\omega_\theta = \left\{ \omega^1, \ldots, \omega^\theta \right\} = \left\{ (y^1, Z^1), \ldots, (y^\theta, Z^\theta) \right\}, \quad \theta = \sum_{j=1}^{p} k(j), \tag{13}$$

полученных всеми процессорами в результате θ завершившихся испытаний.

При этом выбор точки $y^{k+1}(j)$, $k = k(j)$, очередного испытания, которое должно быть реализовано на j-м процессоре, может быть осуществлен с использованием информации ω_θ. Дополнительно может быть учтена также информация о точках $y^{k+1}(i)$, $k = k(i)$, в которых осуществляют испытания другие процессоры ($1 \le i \le p$, $i \ne j$). Множество этих точек обозначим

$$Y_\theta(j) = \bigcup_{i=1, i \ne j}^{p} y^{k(i)+1}(i) \tag{14}$$

В результате, решающее правило для j-го процессора может быть построено в форме функции

$$y^{k+1}(j) = G_{\theta,j}(\omega_\theta, Y_\theta(j)), \quad k = k(j), \quad 1 \le j \le p. \tag{15}$$

Заметим, что, согласно описанной схеме, информация о числе испытаний, осуществленных другими процессорами, анализируется j-м процессором в момент, после завершения очередного испытания. Поэтому, в силу допущения, что моменты выбора точек очередных испытаний различными процессорами не синхронизованы, возможны различия значений θ, зафиксированных различными процессорами в один и тот же момент времени (т.е. $\theta = \theta(j)$). Фактически, предложенная схема интерпретирует результаты испытаний, проведенных другими процессорами, как полученные данным процессором.

3. Возможный способ организации параллельных испытаний в соответствии с предложенной схемой состоит в том, что каждый процессор независимо от других реализует свое решающее правило

для выбора точек испытаний. При этом каждым процессором используется общий информационный массив ω_θ и множество $Y_\theta(j)$, что предполагает обмен информацией между процессорами. Подразумевается, что каждый процессор, завершив испытание, посылает сообщение с результатами вычислений, всем другим процессорам. Кроме того, выбрав некоторую точку для очередного испытания, процессор информирует об этом выборе все остальные процессоры. При наличии указанных обменов *каждый* процессор рождает решение исходной задачи во всей области D (или Q). Следовательно, предлагаемая схема не включает какого-либо выделенного (управляющего) процессора, что повышает надежность функционирования при возможных аппаратных или программных сбоях отдельных процессоров, а также в случае передачи части процессоров для обеспечения других клиентов или других нужд вычислительной системы. В последнем случае (т.е. при уменьшении числа p используемых процессоров) не требуется какого-либо специального информирования об этом изменении значения p.

4. Для иллюстрации изложенного подхода приведем пример решения 5–мерной задачи с 5 ограничениями вида (см., Strongin and Sergeyev (2000)):

$$g_1(w) = -(x + y + z + u + v) \le 0,$$

$$g_2(w) = (y/3)^2 + (u/10)^2 - 1.4 \le 0,$$

$$g_3(w) = 3 - (x+1)^2 - (y+2)^2 - (z-2)^2 - (v+5)^2 \le 0,$$

$$g_4(w) = 4x^2 \sin x + y^2 \cos(y+u) + z^2\left[\sin(z+v) + \sin(10(z-u)/3)\right] - 4 \le 0,$$

$$g_5(w) = x^2 + y^2\left[\sin((x+u)/3 + 6.6) + \sin((y+v)/2 + 9) + 0.9\right]^2 -$$

$$-17\cos^2(z + x + 1) + 16 \le 0,$$

где вектор параметров $w = (x, y, z, u, v)$ принадлежит гиперпараллелепипеду:

$$-3 \le x, y, z \le 3, \quad -10 \le u, v \le 10.$$

Минимизируемая функция является многоэкстремальной и определяется выражением:

$$\varphi(w) = \sin(xz) - (yv + zu)\cos(xy).$$

Равномерная сетка с шагом 0.01 (по каждой координате) содержит порядка $8 \cdot 10^{14}$ узлов. В порядке эксперимента, рассмотренная задача

была решена методом перебора всех узлов такой сетки. Использовалось несколько десятков процессоров. В целях ускорения вычислений итерация в каждом узле начиналась с проверки простейших ограничений (заметим, что нарушение любого ограничения исключает принадлежность узла к точкам допустимой области). Кроме того, простые ограничения были учтены явно путем уменьшения той части области D, в которую помещались узлы сетки. В результате была получена оценка

$$w' = (-0.06,\ 2.20,\ 2.40,\ 9.28,\ 9.63)\,,$$

которой соответствует значение $\varphi(w') = -43.22$. Локальное уточнение этой оценки (с точностью 0.0001 по каждой координате) дало несколько меньшее значение $\varphi^* = -43.2601$, достигаемое в точке

$$w^* = (0.0521,\ 2.2041,\ 2.3911,\ 9.2747,\ 9.6389)\,.$$

В целях иллюстрации, на рис. 5 показан пример двухмерного сечения области D, проходящего через полученную точку w^* (образ этой точки отмечен темным кружком). Рисунок соответствует xy-сечению, т.е. он является образом квадрата, содержащего все точки

$$w = (x,\ y,\ 2.391,\ 9.275,\ 9.639), \quad -3 \le x, y \le 3.$$

Линии, изображенные на этом рисунке, снабжены цифрами 2, 4, 5, нанесенными в окрестностях границ, соответствующих ограничениям с теми же номерами (каждая цифра помещена в части области, точки которой удовлетворяют соответствующему ограничению).

Далее задача была редуцирована к 10 связанным задачам оптимизации; для их одновременного решения был использован параллельный многомерный индексный метод для вычислений в многопроцессорной системе с 10 процессорами. Общее число итераций составило 59697. Полученная оценка

$$w'' = (-0.068,\ 1.962,\ 3.431,\ 9.833,\ 9.833)$$

характеризуется значением $\varphi(w'') = -42.992$, которое улучшилось (после локального уточнения решения) до значения $\varphi(w^{**}) = -43.2985$. Заметим, что в рассмотренном примере экономия итераций (за счет использования неоднородных сеток) составляет много порядков, что и оправдывает разработку и использование сложных решающих правил алгоритмов глобального поиска.

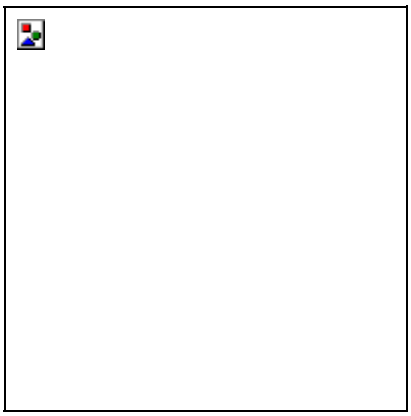


Рис. 5. Линии уровня минимизируемой функции и границы, соответствующие ограничениям для плоского двухмерного сечения, проходящего через точку w^*

Общая характеристика эффективных способов представления поисковой информации

1. Схема распараллеливания, рассмотренная в работе и предназначенная для решения задач вида (1)–(3) на вычислительных системах кластерного типа, предусматривает, что каждый из p используемых процессоров характеризуется уникальным номером j, $1 \le j \le p$, и имеет:

- программный модуль для вычисления значений триад (11) в задаваемых точках $y \in D$;
- систему управления базой данных для хранения информации, представленной множествами ω_θ и $Y_\theta(j)$ соответственно из (13) и (14);
- средства для осуществления информационных обменов между процессорами;
- программный модуль, реализующий правило (15) выбора точек испытаний $y^{k+1}(j)$, по информации, представленной в указанной базе данных.

Примем, что все (возможно не одинаковые) процессоры являются равноправными (т.е. нет выделенного управляющего процессора) и не синхронизируют свою работу. Последнее означает, что для

осуществления выбора точки очередной итерации процессор не должен ожидать сообщений от других процессоров. Как уже отмечалось, такая схема хорошо согласуется с тем обстоятельством, что время, необходимое для выполнения испытания в некоторой точке гиперинтервала D, может существенно зависеть от этой точки.

Теперь остановимся на проблеме информационных обменов между процессорами, которые должны (для каждого процессора) обеспечивать пополнение упомянутых выше баз данных сведениями о триадах, оцененных другими процессорами (что обеспечивает формирование соответствующих множеств ω_θ). Кроме того, каждый процессор должен получать информацию о точках, в которых другие процессоры начали осуществлять испытания (на момент, когда процессор–получатель завершил вычисление очередной триады и анализирует информацию для выбора точки следующего испытания).

Пусть кластер включает p процессоров. Примем, что передача информации от одного процессора к другому осуществляется путем засылки сообщения от процессора–источника в специальную очередь, созданную на процессоре–получателе. При этом на каждом процессоре создаются p очередей. Очередь, созданная на процессоре с номером l для приема сообщений от процессора с номером j будем обозначать как Q_{jl}, $1 \le l \le p$ (разумеется, можно ограничиться единственной очередью, создаваемой на каждом процессоре для приема сообщений от всех процессоров – в этом случае сообщения от каждого процессора должны содержать его идентификатор). Рис. 6 иллюстрирует описанную выше схему взаимосвязи процессоров с использованием очередей.

В процессе оптимизации каждый процессор с номером j, завершив очередное испытание, засылает полученное значение триады (11) во все «свои» (т.е. имеющие первый индекс, равный j) очереди на других процессорах, а также очередь Q_{jj}, созданную на нем самом. Это последнее предложение устанавливает единый источник получения данных для формирования множества ω_θ из (13). Все триады, составляющие множество (13) и представленные в базах данных процессоров, получены из очередей, созданных на соответствующих процессорах.

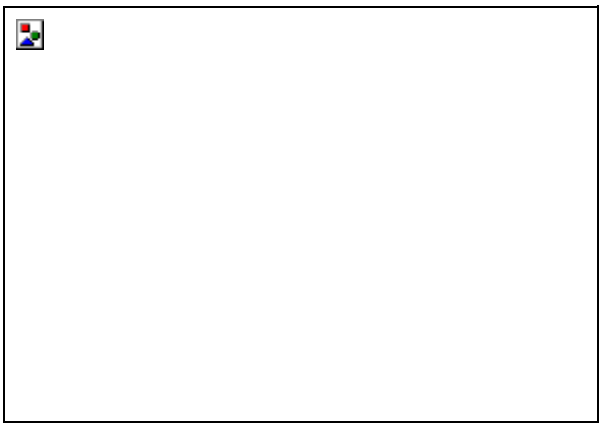


Рис. 6. Схема информационных связей между процессорами

Помимо пересылки полученных значений триад, процессоры должны также информировать друг друга о точках, в которых они начали очередное испытание. Чтобы унифицировать формат передаваемых сообщений условимся записывать информацию о точке очередного испытания в виде «неполной» триады. Первой компонентой такой триады является точка $y^{k+1}(j)$, очередного испытания. Вторая компонента остается неопределенной, ибо оценка значения вектор функции $F(y)$ в указанной точке $y^{k+1}(j)$, еще не завершена. Третья компонента содержит отрицательное значение индекса (условимся, например, что это значение есть $\nu = -1$). Поскольку значения индекса в точках гиперинтервала D не могут быть отрицательными, то введенное соглашение позволяет идентифицировать неполные пары, содержащие информацию о точках, в которых осуществляются итерации.

С учетом принятых соглашений, вернемся к рассмотрению схемы обменов сообщениями. Завершив очередное испытание и заслав полученное значение триады в «свои» очереди на всех процессорах, процессор с номером j последовательно извлекает информацию из всех созданных на нем очередей и включает соответствующие триады во множество ω_θ, содержащееся в его базе данных. Обнаруживаемая неполная триада либо сопровождается полной триадой, имеющей ту же первую компоненту (т.е. ту же точку проведения испытания), либо

она является последним элементом соответствующей очереди. Любой другой случай является следствием ошибки и следует предусматривать средства реагирования на такие ошибки. Неполная триада, сопровождаемая полной триадой, исключается из дальнейшего рассмотрения, ибо процессор, посылавший эту информацию о точке очередного испытания, уже завершил это испытание и переслал триаду-результат. Вектор, представляющий точку испытания из неполной триады, завершающей некоторую очередь, включается в множество $Y_0(j)$ из (14). Поскольку любой процессор считывает содержание очередей после отсылки определенной им триады, то информационные массивы, извлекаемые из очередей с совпадающими индексами, не могут завершаться неполными триадами.

2. Конкретные задачи глобальной оптимизации, возникающие в приложениях, обычно являются многомерными. Это обстоятельство затрудняет построение экономных неравномерных сеток с помощью правил последовательного выбора узлов, использующих информацию, накапливаемую в процессе решения задачи. Дело в том, что правила вида (7) реализуют оптимальный выбор путем решения некоторой вспомогательной оптимизационной задачи. И поскольку выбираемая точка y^{k+1} принадлежит многомерному гиперинтервалу D из (3), то реализация правила (7) предполагает решение многомерной оптимизационной задачи, характер которой усложняется с каждым новым испытанием (ввиду увеличения числа величин, являющихся аргументами функции G_k из (7)). Т.е. задача выбора точки очередного испытания сама оказывается задачей типа (1)–(3), решаемой на каждом шаге процесса и усложняющейся от шага к шагу.

Возможный путь преодоления этих трудностей состоит в редуцировании многомерных задач, определенных в гиперинтервале D из (3), к эквивалентным им одномерным задачам. Основная идея использования таких редукций связана с возможностью разработки эффективных методов оптимизации сложных одномерных функций.

Возможная схема такой редукции состоит в следующем (см., например, Стронгин (1978), Strongin and Sergeyev (2000)). Введем «стандартный» гиперкуб с единичным ребром D^* вида

$$D^* = \left\{ w \in R^N : -2^{-1} \le w_j \le 2^{-1}, 1 \le j \le N \right\} \qquad (16)$$

и сведем задачу, заданную на гиперинтервале D, к задаче, заданной на стандартном кубе D^* из (16). Такое сведение обеспечивается

следующим простым линейным преобразованием координат (с одинаковым коэффициентом растяжения по всем координатным осям):

$$w_j = \left(y_j - \left(a_j + b_j\right)/2\right)/\rho, \quad 1 \le j \le N, \tag{17}$$

где $\rho = \max\left\{b_j - a_j : 1 \le j \le N\right\}$.

Кроме того, вводится дополнительное ограничение вида

$$g_0(w) = \max\left\{\left|w_j\right| - \left(b_j - a_j\right)/2\rho : 1 \le j \le N\right\} \le 0\,, \tag{18}$$

позволяющее определить принадлежность образа (при отображении обратном (17)) точки $w \in D^*$ к гиперинтервалу D.

Следующий шаг состоит в использовании непрерывного однозначного соответствия $w(x)$ типа кривой Пеано для отображения единичного отрезка $[0,1]$ вещественной оси x на стандартный гиперкуб D^* из (16). При этом

$$D^* = \left\{ w(x) : x \in [0,1] \right\} \tag{19}$$

и, поскольку отображение $w(x)$ является непрерывным, то

$$\min\left\{\varphi(w) : w \in D^*,\, g_i(w) \le 0,\, 1 \le i \le m\right\} =$$
$$= \min\left\{\varphi(w(x)) : x \in [0,1],\, g_i(w(x)) \le 0,\, 0 \le i \le m\right\}.$$

Аналогично обеспечивается сведение и других многомерных задач, рассмотренных в работе, к эквивалентным одномерным задачам.

Важным свойством предложенной схемы редукции является сохранение ограниченности разностей значений липшицевых функции при ограниченных изменениях значений их аргументов. Если исходная функция $F_i(w)$, $1 \le i \le s$, удовлетворяет условию Липшица с некоторой константой L_i, то соответствующая ей одномерная функция $F_i(w(x))$, удовлетворяет равномерному условию Гёльдера с показателем N^{-1} и коэффициентом K_i:

$$\left|F_i(w(x')) - F_i(w(x''))\right| \le K_i\left(\left|x' - x''\right|\right)^{1/N}, \quad x', x'' \in [0,1], \tag{20}$$

где $K_i = 2L_i\sqrt{N+3}$. Заметим, что при использовании разверток, не являющихся кривыми, заполняющими пространство (этот термин также часто используется для обозначения кривых из класса кривых Пеано), свойство ограниченности разностей аналогичное (20) может не иметь места. В частности, этим свойством не обладают простые развертки типа «телевизионных» или спиралей.

Свойство (20) обеспечивает оценки значений искомых величин, характеризующих многомерную область, путем использования результатов испытаний в узлах сетки, погруженной в единичный отрезок. При этом сохраняется также возможность последовательного построения эффективных неравномерных сеток.

Численные методы для приближения кривых Пеано (с любой заданной точностью), а также методы построения обратных отображений (которые являются кратными) описаны и обоснованы в работах Стронгин (1978,1990), а также в Strongin and Sergeyev (2000). Последняя работа содержит также программы на языке *C*++.

Описанная схема редукции к одномерным задачам позволяет построить унифицированную базу данных (ω_θ, $Y_\theta(j)$), в которой вместо точек испытаний $w^l \in D^*$ используются их прообразы $x^l \in [0,1]$ при соответствии $w(x)$. Т.е. в каждой триаде записывается вещественный индекс x^l, обеспечивающий возможность упорядочения всех триад в соответствии со значениями этого индекса. При этом вставка новых триад производится таким образом, чтобы сохранялось упорядочение, предписанное индексом. Возможный вариант организации ведения такой базы описан в работе Стронгин, Гергель (1978). Важно также отметить, что упорядочение всех данных на одномерной шкале упрощает описание и реализацию решающих правил для выбора точек испытаний. Правилами определяется очередной узел $x^{k+1} \in [0,1]$, который затем отображается в точку $w^{k+1} = w(x^{k+1})$.

Следует отметить также, что в силу свойств кривых Пеано, двум близким точкам x^l, $x^t \in [0,1]$ будут соответствовать их близкие образы из многомерной области D

$$w^l = w(x^l), \quad w^t = w(x^t).$$

Обратное утверждение может, однако, оказаться неверным, т.е. конкретные прообразы x^l, x^t двух близких точек w^l, w^t могут оказаться не близкими. В работах Стронгин (1978,1990) и Strongin and Sergeyev (2000) предлагаются пути преодоления этого недостатка.

Следует обратить внимание на общий характер этой проблемы, поскольку многие другие известные методы решения многомерных задач также используют то или иное сведение к одномерным задачам (или к семействам одномерных задач). К числу схем такого редуцирования относится, например, интегрирование по схеме кратного интеграла:

$$\int_D \varphi(y)dy = \int_{a_1}^{b_1} \dots \int_{a_N}^{b_N} \varphi(y_1,\dots,y_N)dy_1 \dots dy_N \ .$$

Аналогичную роль играет сведение задачи минимизации в гиперинтервале D к решению семейства «вложенных» одномерных задач:

$$\min\{\varphi(y) : y \in D\} = \min_{a_1 \le y_1 \le b_1} \dots \min_{a_N \le y_N \le b_N} \varphi(y_1,\dots,y_N) \ . \qquad (21)$$

Такое редуцирование ведет к большей потере информации о близости точек в многомерном пространстве, чем рассмотренное выше использование разверток типа кривой Пеано. Для иллюстрации вернемся к приведенному в работе примеру 2. На рис. 2 показано расположение 100 точек испытаний при минимизации конкретной тестовой функции из семейства (9) с помощью метода, использующего развертку типа кривой Пеано. При этом

$$\min\{\varphi(y) : y \in D\} = \min\{\varphi(y(x)) : x \in [0,1]\} \ .$$

Напомним, что в этом случае область D представляет собой квадрат с единичной стороной и, поэтому, нет необходимости в дополнительном ограничении типа (18).

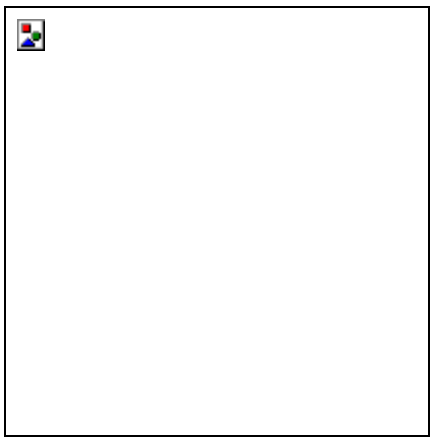


Рис. 7. Линии уровня тестовой функции из семейства (9) и точки выполнения 270 испытаний

Теперь рассмотрим результаты решения этой же задачи по многошаговой схеме (21). Расположение точек 270 испытаний, выполненных в соответствии с этим подходом (одномерные подзадачи решались информационным алгоритмом Стронгина), отмечены темными прямоугольниками на рис. 7. Рисунок наглядно показывает наличие сгущений узлов сетки вдали от точки глобального оптимума, что является недостатком схемы (21). Пример взят из работы Strongin and Sergeyev (2000).

3. Поисковая информация ω_θ из (13), получая в процессе вычислений, при использования отображения $w(x)$ из (19) может быть преобразована к виду

$$\Omega_k = \left\{\alpha^1,...,\alpha^k\right\}, \tag{22}$$

в котором каждая триада $\omega^i = (y^i, Z^i, \nu^i)$, $1 \le i \le k$, представленным редуцированным аналогом $\alpha^i = (y^i, Z^i, \nu^i)$ при соответствии $y^i = w(x^i)$. Для эффективного выполнения вычислительных процедур обработки поисковой информации расположение редуцированных триад целесообразно осуществлять в порядке возрастания одномерных индексов x^i, $1 \le i \le k$, т.е. множество Ω_k из (22) приводится к виду

$$A_k = \left\{\alpha_1...,\alpha_k\right\}, \ x_1 < ... < x_k \ . \tag{23}$$

Следует отметить, что введенная при помощи нижнего индекса нумерация триад в порядке возрастания одномерных прообразов x^i, $1 \le i \le k$, является относительной и должна переустанавливаться при каждом изменение состава поисковой информации Ω_k.

Проблема накопления и обеспечения упорядоченного представления поисковой информации в виде (23) является одной из самых значительных при численной реализации алгоритмов решения рассмотренного в работе класса задач. Принципиальные моменты при разрешении данной проблемы состоят в организации хранения информации большого объема ($k \gg 1$), обеспечения упорядоченного представления этой информации и организации эффективной обработки. Исследования эффективных способов представления поисковой информации проводились на протяжении длительного ряда лет – см., например, Стронгин, Гергель (1978), Gergel and Sergeyev (1999) и др.

4. Возможный подход для эффективной организации поисковых данных состоит в представлении упорядоченного множества A_k из (23) в виде последовательного набора фрагментов (*блоков*)

$$A_k \sim B_q = [\beta_1 ..., \beta_q], \tag{24}$$

где каждый блок β_j, $1 \le j \le q$, набора B_q представляет собой группу последовательных триад множества A_k

$$\beta_j = [\alpha_{i_j+1} ..., \alpha_{i_j+k_j}], \tag{25}$$

и для каждой пары соседних блоков β_j, β_{j+1}, $1 \le j \le k$, должны выполняться правила

- $i_{j+1} = i_j + k_j$ – формирование блоков осуществляется последовательно по триадам множества A_k (блок β_{j+1} следует непосредственно за блоком β_j); при этом $i_1 = 1$, $i_q + k_q = k$;
- $\beta_j \cap \beta_{j+1} = \{\alpha_{i_j+k_j}\}$ – блоки пересекаются по граничным триадам; каждая триада множества A_k принадлежит только одному блоку набора B_q (за исключением граничным триад блоков, которые содержатся в двух блоках).

Для определения соответствия множества A_k из (23) и блочного набора B_q из (24) необходимо наличие описания структуры разбиения, которое может быть организовано в виде каталога

$$H_q = [\eta_1 ..., \eta_q],$$

в котором для каждого блока β_j, $1 \le j \le q$, из (25) запоминаются параметры $\eta_j = (k_j, X_j)$, где k_j – количество триад в блоке, а X_j – значение максимального одномерного прообраза в блоке.

Блочное представление множества A_k в значительной степени решает вопросы хранения и упорядочения поисковой информации:

- блочность данных позволяет разместить информацию во внешней памяти прямого доступа (магнитных дисках); для обработки блоки могут перемещаться в оперативную память;
- при блочном представлении операция упорядочения множества A_k может быть сведена к локальным операциям упорядочивания отдельных блоков; при выборе блоков достаточно малого размера выполнение операции сортировки не будет приводить к большим вычислительным затратам.

5. Для организации хранения блоков может быть использован страничный принцип представления памяти, когда вся используемая

для вычислений внешняя и оперативная память формируется в виде областей одинакового размера (страниц). Использование такого подхода приводит к необходимости разрешения всех общих системных вопросов, связанных с организацией страничной виртуальной памяти (планирование размещения блоков данных, решение вопросов подкачки, выбор стратегий замещения и т.п.).

Для проведения рассмотренных экспериментов использовалось оборудование компаний Intel и Hewlett Packard, переданное в Нижегородский госуниверситет в качестве грантов для образовательных и научно-исследовательных целей. Исследования способов организации параллельных вычислений в задачах выбора глобально-оптимальных решений для многопроцессорных кластерных систем, выполнялись в рамках работ, поддержанных грантом компании Intel.

Литература

1. Гергель В.П., Стронгин Р.Г. (2000). Параллельная глобальная оптимизация в многопроцессорных вычислительных системах. // Труды Всероссийской научной конференции "Высокопроизводительные вычисления и их приложения". М.: Изд-во МГУ. С. 87-92.
2. Гришагин В.А. (1978). Операционные характеристики для некоторых алгоритмов глобального поиска. // Проблемы статистической оптимизации. Рига: Зинатне. С.198-206.
3. Пиявский С. А. (1972). Один алгоритм отыскания абсолютного экстремума функции // Ж. вычисл. матем. и матем. физ. Т. 12. № 4. С. 888-897.
4. Стронгин Р.Г. (1978). Численные методы в многоэкстремальных задачах. М.: Наука.
5. Стронгин Р.Г., Гергель В.П., (1978). Реализация на ЭВМ обобщенного многомерного алгоритма глобального поиска. // Проблемы кибернетики. С. 59-66.
6. Стронгин Р.Г., Маркин Д.Л. (1986). Минимизация многоэкстремальных функций при невыпуклых ограничениях // Кибернетика, № 4. С.64-69.

7. Стронгин Р.Г., Сергеев Я.Д. (1989). Алгоритмы глобальной минимизации с параллельными итерациями // Ж. вычисл. матем. и матем. физ., Т. 35, № 5. С.705-717.
8. Стронгин Р.Г. (1990). Поиск глобального минимума. М.: Знание.
9. Gergel V.P., Sergeyev Ya.D. (1999) Sequential and parallel global optimization algorithms using derivatives. Computers & Mathematics with Applications, 37(4/5), 163-180.
10. Kushner, H.J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. Transactions of ASME, Ser. D. Journal of Basic Engineering 86, 1, 97-106.
11. Pfister, G. P. (1995). In Search of Clusters. Prentice Hall PTR, Upper Saddle River, NJ (2nd edn., 1998).
12. Strongin R.G., Sergeyev Ya.D. (2000). Global optimization with non-convex constraints: Sequential and parallel algorithms. Kluwer Academic Publishers, Dordrecht

ИСПОЛЬЗОВАНИЕ ТЕТРАЭДРИЧЕСКИХ СЕТОК ДЛЯ МОДЕЛИРОВАНИЯ ОБТЕКАНИЯ ТЕЛ НА МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ*

С.А. Суков

Институт математического моделирования РАН, г.Москва

В настоящее время широкое распространение получает моделирование физических процессов на многопроцессорных системах с использованием неструктурированных тетраэдрических сеток. Тетраэдрические сетки, содержащие достаточно большое число узлов, с высокой точностью аппроксимируют пространство около поверхности трехмерных тел сложной формы, но требуют при этом больших вычислительных затрат. В данный момент соответствующую производительность предоставляют многопроцессорные системы, преимущественно с распределенной памятью, которые позволяют практически неограниченно наращивать вычислительную мощность и обрабатывать большие объемы информации.

* Работа выполнена при поддержке РФФИ (гранты № 02-01-00589, 02-01-00700, 03-01-06108).

В данной работе рассматривается система уравнений Эйлера, описывающая течение невязкого совершенного газа в трехмерной пространственной геометрии. Пространственная дискретизация уравнений Эйлера проводится на неструктурированных тетраэдрических сетках с использованием метода конечного объёма. Значения всех газодинамических переменных определяются в вершинах тетраэдров. Внутри каждого тетраэдра вокруг его вершин строится контрольный объем (расчетная ячейка) по следующим точкам: центр тяжести тетраэдра, центры тяжести граней и середины ребер тетраэдра [1]. Базовая аппроксимация первого порядка конвективных членов уравнения Эйлера или численный поток через грани контрольного объема строится на основе решения задачи Римана по направлению внешней нормали к грани с использованием двух различных методов. Первый метод базируется на приближенном решении задачи Римана – схемы Ошера и Роу. Второй метод – это метод расщепления вектора потока (рассматривается его частный случай, так называемая кинетически согласованная разностная схема или кинетическое расщепление). Повышение порядка аппроксимации достигается заменой кусочно-постоянного распределения газодинамических параметров в расчетной ячейке на кусочно-линейное распределение (аппроксимация типа MUSCL). При определении кусочно-линейного распределения используются разностные градиенты газодинамических параметров двух типов: нодальный или эрмитов градиент, определенный в узле и рассчитанный по контрольному объему окружающий этот узел и градиент, распределенный на тетраэдре (градиент от линейной функции на тетраэдре). Определенный таким образом численный поток повышенного порядка (до третьего порядка в линейном приближении), может приводить к появлению нефизических осцилляций разностного решения в окрестности разрывных течений (ударных волн). В этом случае в окрестностях разрывов и больших градиентов делается локальное переключение на схему первого порядка в зависимости от знака соседних разностных градиентов. Интегрирование по времени осуществляется явным линейным методом типа Рунге-Кутта второго порядка (метод Джеймсона).

Как видно из описанной выше модели, основная вычислительная нагрузка во время расчетов приходится на определение численных

потоков через грани контрольных объемов. При построении контрольных объемов приведенным способом две соседние расчетные ячейки имеют несколько общих граней, число которых равно числу тетраэдров, опирающихся на ребро, соединяющее центры ячеек. Для снижения вычислительных затрат выполняется суммирование граней, т.е. ячейки приводятся к такому виду, что два соседних контрольных объема имеют только одну общую грань.

Балансировка загрузки процессоров выполняется с использованием параллельного алгоритма иерархического разбиения графов, основанного на объединении вершин в пары [2]. Он позволяет разбить сетку на компактные примерно равные по размеру подобласти (микродомены), минимизировав при этом число разрезанных в результате разбиения ребер. В итоге каждый процессор получает описание (список вершин) микродомена, обрабатываемого на нем в процессе моделирования. Такой подход к балансировке загрузки процессоров позволяет добиться высокой эффективности распараллеливания как для систем с распределенной, так и с общей памятью.

Несмотря на высокую эффективность параллельного алгоритма балансировки загрузки с ростом числа узлов сетки процесс ее разбиения на подобласти все равно требует значительных затрат времени. Кроме того, возникают специфические для многопроцессорных систем проблемы, негативно влияющие на эффективность использования вычислительного комплекса. Резко возрастают затраты времени на чтение и запись данных о сетке и сеточных функциях, вычисление необходимых для численного моделирования геометрических параметров сетки и определение информации, которой процессоры будут обмениваться во время моделирования. Эти затраты становятся сопоставимыми со временем отдельного сеанса моделирования. В результате моделирование с использованием достаточно крупных сеток становится неэффективным. В этом случае для работы с сетками используются алгоритмы двухуровневого распределенного хранения и ввода-вывода данных о сетке и значений сеточных функций [3]. Перед началом серии расчетов выполняется предварительная обработка сетки. На первом этапе предварительной обработки исходная сетка делится на достаточно большое число микродоменов, после чего в соответствии с

полученным разбиением выполняется перенумерация ее улов. Одновременно формируется общее описание структуры сетки (граф микродоменов). На этапе балансировки загрузки именно граф микродоменов делится на подобласти, число которых равно числу выполняющих моделирование процессоров. В результате каждый процессор получает список обрабатываемых на нем микродоменов. На втором этапе для каждого микродомена сетки собирается информация необходимая для автономного определения параметров контрольных объемов, построенных вокруг его вершин. После этого сетка записывается в виде набора файлов, содержащих описание графа микродоменов, множество описаний микродоменов, таблицу соответствия старых и новых номеров узлов сетки и таблицу размещения микродоменов по файлам. Распределенное хранение значений сеточных функций в узлах микродоменов осуществляется аналогичным способом. Данные могут записываться как в сжатом, так и неупакованном виде.

Использование приведенных алгоритмов позволяет повысить эффективность использования вычислительного комплекса и обрабатывать сетки, размер которых ограничивается только объемом доступной оперативной памяти системы.

Литература

1. Barth T.J. Aspects of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations. AGARD-VKI Lecture Series, AGARD R787.
2. Abalakin I.V., Boldyrev S.N., Chetverushkin B.N., Iakobovski M.V., Zhokhova A.V. Parallel Algorithm for Solving Problems on Unstructured Meshes. 16th IMACS World Congress 2000 Proceedings. Lausanne, August 21-25, 2000.
3. Суков С.А., Якобовский М.В. Обработка трехмерных неструктурированных сеток на многопроцессорных системах с распределенной памятью. Фундаментальные физико-математические проблемы и моделирование технико-технологических систем: Сборник научных трудов / Под редакцией Л.А. Уваровой. – М.: Издательство «Янус-К», 2003.

ПРОГРАММНАЯ СИСТЕМА ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В ЗАДАЧАХ ВЫБОРА ГЛОБАЛЬНО-ОПТИМАЛЬНЫХ РЕШЕНИЙ

А.В. Сысоев

Нижегородский государственный университет им. Н.И.Лобачевского

В настоящей работе представлено краткое описание программной системы параллельных вычислений в задачах выбора глобально-оптимальных решений под рабочим названием «Абсолют Эксперт». Программная система «Абсолют Эксперт» позволяет ставить и решать (исследовать) задачи выбора оптимального варианта, приводимые к постановке, сокращенное описание которой указано в разделе «Класс задач».

В силу существенной вычислительной сложности задач, на решение которых ориентирована система, обеспечивается возможность ее функционирования в распределенном режиме на нескольких вычислительных узлах. В архитектуре системы выделяются: базовый уровень – основа системы – представляет собой библиотеку структур данных и функций, содержит всю необходимую функциональность системы; прикладной уровень – надстройка над базовым уровнем – обеспечивает более удобные средства работы конечного пользователя с системой.

Класс задач

I. Математическая модель объекта

Пусть объект исследования характеризуется набором параметров $\bar{y} = (y, u)(y_1, \ldots, y_M, u_1, \ldots, u_M)$и вектор-функцией характеристик $w(\bar{y})=(w_1(\bar{y}),\ldots, w_n(\bar{y}))$, определенных таким образом, что уменьшение значений характеристик соответствует лучшему выбору.

Предполагается, что:

1) Координаты вектора $y = (y_1, \ldots, y_N)$ могут непрерывно изменяться в гиперинтервале $D = \{y \in R^N: a_i \le y_i \le b_i, 1 \le i \le N\}$.

2) Кортеж u принимает значения в виде набора дискретных параметров из заданного множества

$$U = U_1 \times \ldots \times U_M,\ U_i = \left\{ u_{i1}, \ldots, u_{ik_i} \right\},\ 1 \le i \le M.$$

3) Каждая характеристика $w_i(y, u(\tau)),\ 1 \le i \le n,\ 1 \le \tau \le s$ является липшицевой функцией с соответствующей константой Липшица $K_{i\tau}$, которая может быть не задана.

II. Функциональные ограничения

В отношении части координатных функций w_j, номера которых определяются множеством $G = \{j_1, \ldots, j_m\} \subset \{1, \ldots, n\}$, ставится условие уменьшения их значений до некоторых заданных допусков q_i.

III. Векторный критерий эффективности

Часть координатных функций w_i, номера которых определяются множеством $F = \{i_1, \ldots, i_k\} \subset \{1, \ldots, n\}$, рассматривается как векторный критерий эффективности

$$f(y,\tau) = \left(f_1(y,\tau), \ldots, f_k(y,\tau)\right),\ f_j(y,\tau) = w_{i_j}\left(y, u(\tau)\right),\ i_j \in F.$$

Допускается, что на некоторую характеристику w_i накладывается функциональное ограничение, т.е. требование непревышения заданного допуска q_i, и одновременно ставится задача рассмотрения возможности дополнительного уменьшения значения этой характеристики (т.е. она включается в векторный критерий как одна из координат). При этом множества G и F будут иметь непустое пересечение.

IV. Понятие оптимального решения

Оптимальный выбор в рассматриваемом классе задач основан на предполагаемой упорядоченности частных критериев f_j, составляющих векторный критерий, по важности: главным критерием является f_1, менее важен f_2, затем следуют $f_3, f_4, \ldots, f_k$. Тогда оптимальный выбор определяется методом уступок. Его результатом для заданного набора уступок ε_i, $1 \le i \le k$ будет пара (y^*, τ^*), которая является решением рекурсивной задачи:

$$\left(y^*, \tau^*\right) = \arg \min_{1 \le \tau \le s} \min_{y \in Q_{m+1}(\tau)} \left\{ f_k(y,\tau) :\ f_j(y,\tau) \le f_j^* + \varepsilon_j,\ 1 \le j \le k-1 \right\},$$

$$f_i^* = \min_{1 \le \tau \le s} \left(\min \left\{ f_i(y,\tau),\ y \in Q_{m+1}(\tau) :\ f_j(y,\tau) \le f_j^* + \varepsilon_j,\ 1 \le j \le i-1 \right\} \right),$$

$$1 \le i \le k$$

Метод оценки оптимального выбора

В системе реализована следующая схема оценки оптимального выбора исходной многомерной многокритериальной задачи с упорядоченными критериями и существенно нелинейными, невыпуклыми ограничениями, зависящей также и от дискретных параметров:

1. Проводится последовательное редуцирование исходной задачи к семейству скалярных одномерных, однокритериальных задач.

2. Для сохранения и учета информации о близости точек в многомерном пространстве в ходе выполнения редукции используется подход, приводящий к построению множественной развертки [2]. В результате такого построения формируется семейство из $L+1$ одномерной задачи оптимизации, каждая из которых эквивалентна исходной N-мерной постановке и определена на интервале [0,1].
3. Для полученного семейства выполняется свертка в одну задачу, определенную на интервале $[0, L+1]$.
4. Выполняется еще одна свертка, сводящая операцию взятия минимума по целочисленному параметру τ, $1 \le \tau \le s$ к операции минимизации по непрерывному параметру x. Как следствие область определения новой задачи составит интервал $[0, (L+1)s]$.

Полученная в результате описанных выше преобразований одномерная скалярная многоэкстремальная задача с невыпуклыми ограничениями и заданными точками разрыва первого рода (далее стандартная задача) решается индексным методом.

Параллельная схема вычислений

Необходимо отметить, что индексный метод решения стандартной задачи позволяет результаты любого испытания (определение очередной точки итерации x_i и вычисление функционалов задачи), выполненного в одном из подынтервалов вида $[is, (i+1)s]$, $0 \le i \le L$, интерпретировать как результаты одновременно выполненных $L+1$ испытаний во всех подынтервалах указанного вида. Этот факт позволяет осуществить модификацию индексного метода для выполнения расчетов на нескольких вычислительных узлах.

Важным свойством рассматриваемого класса задач в совокупности с предлагаемой схемой решения является то, что времена выполнения испытаний могут существенно различаться как от точки к точке внутри одного подынтервала, так и между подынтервалами. Вследствие этого эффективная схема параллельного решения должна предполагать асинхронность выполнения расчетов отдельными вычислительными узлами, а значит допускать разнотипность узлов и возможность их отличия по вычислительным характеристикам.

Реализованный в системе вариант такой эффективной схемы подробно описан например в [1].

Программная реализация

Функциональность программной системы «Абсолют Эксперт» обеспечивается следующими подсистемами:

- подсистема Оптимизация – отвечает за построение объекта оптимизации, постановку задач, определение и настройку методов, назначение заданий и выполнение процесса поиска оптимального варианта в рамках изложенной выше модели;
- подсистема Поисковая информация – реализует структуры данных и методы, обеспечивающие получение, хранение и анализ информации, генерируемой в ходе итераций поиска;
- подсистема Страничная память – содержит структуры данных и методы для организации страничного представления оперативной памяти, используемой для обработки поисковых данных;
- подсистема Архив – предназначена для организации хранения поисковой информации во внешней памяти;
- подсистема Очередь характеристик – ускоряет работу алгоритмов при больших объемах поисковой информации (случай высокой точности вычислений или большая размерность исходной постановки);
- подсистема Обработка состояний – имеет целью создание в рамках программного комплекса унифицированной схемы контроля и наблюдения за состояниями системы в целом.

Текущая версия системы поддерживает постановки задачи, содержащие до 10 функционалов (ограничений и критериев), определенных на гиперинтервале пространства R^N, где $N \le 10$; предоставляет возможность приостановки и продолжения счета, загрузки функционалов из dll-файла. В составе прикладного уровня текущей версии системы ведется реализация интерфейса в виде оконного приложения.

Литература

1. Гергель В.П., Стронгин Р.Г. (2001). Параллельные вычисления в задачах выбора глобально-оптимальных решений для многопроцессорных кластерных систем. // Современные методы математического моделирования. Сборник лекций Всероссийской молодежной школы международной конференции «Математическое моделирование». Самара. С. 46-55.

2. Strongin R.G., Sergeev Ya.D. (2000). Global optimization with non-convex constraints: Sequential and parallel algorithms. Kluwer Academic Publisher, Dordrecht.

РАЗРАБОТКА ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ ОБХОДА ДЕРЕВА

Н.Е. Тимошевская

Томский государственный университет

Введение

Существует большой класс сложных вычислительных задач, в том числе комбинаторно-логических, решение которых осуществляется путем последовательного просмотра (перечисления) элементов конечного множества, содержащего искомые объекты, и выбора среди них подходящих. Обеспечить такой просмотр можно с помощью методов, использующих дерево поиска решений. Предлагаются параллельные методы обхода дерева поиска, позволяющие наиболее эффективно использовать ресурсы кластерной системы, и за счет этого ускорить процесс решения задачи. Приводится описание генераторов случайных деревьев и деревьев поиска сочетаний и оптимального назначения, созданных для экспериментального анализа предложенных методов.

1. Последовательные обходы дерева

Выполнить обход дерева значит в некотором порядке посетить каждый узел дерева. Посетить узел может означать, например, следующее: обработать некоторую информацию, сопоставленную этому узлу, выполнить некоторую процедуру, удалить узел и т.п.

В зависимости от порядка посещения (прохождения) узлов определяют различные виды обходов дерева, среди них обход в глубину и обход в ширину. При обходе в глубину в каждом поддереве вначале проходится корень, а затем его поддеревья. Обход в глубину удобно описывается и реализуется с помощью рекурсивной процедуры, но она не применима для деревьев с большим числом узлов. В нерекурсивном алгоритме обхода в глубину используется стек для упорядочивания обхода, узлы посещаются по мере их извлечения из стека [1].

При обходе в ширину вначале проходится корень дерева – вершина нулевого яруса, затем все вершины первого яруса, затем все вершины второго яруса и т.д. так, что, никакая вершина некоторого яруса не может быть пройдена, пока не будут пройдены все вершины предыдущего яруса. В алгоритме *обхода дерева в ширину* для упорядочивания прохождения узлов используется очередь. Обход в ширину может быть использован для обхода части дерева (начальных ярусов) и построения списка, содержащего заданное число вершин. В результате все вершины, прошедшие через очередь, оказываются пройденными, и вершины, оставшиеся в очереди, принадлежат последнему пройденному и следующему за ним ярусам (в соответствии с порядком обхода в ширину).

2. Параллельные алгоритмы обхода

Ставится задача построения эффективного параллельного алгоритма обхода дерева. *Эффективность алгоритма* определяется как отношение ускорения, достигаемого системой при выполнении данного алгоритма, к числу устройств. Для многопроцессорных систем, состоящих из одинаковых устройств, понятие эффективности совпадает с загруженностью [2]. В этом случае, ставится задача построения параллельных алгоритмов обхода дерева, обеспечивающих наибольшую загруженность системы.

2.1. Метод выделяемых поддеревьев

В основе предлагаемого метода лежит принцип выделения поддеревьев. На первом этапе алгоритма выполняется обход дерева, начиная от его корня, до тех пор, пока число вершин m на некотором ярусе с номером i не достигнет числа параллельных процессов s. Для этого используется алгоритм обхода в ширину, в результате работы которого образуется множество попарно не смежных вершин, которые являются корневыми вершинами поддеревьев. С той же целью может быть также использован рекурсивный алгоритм. Операции данного этапа выполняются по обычному последовательному алгоритму на одном из процессоров системы. Далее для удобства изложения мы будем говорить о системе параллельных процессов. Каждый процесс есть программа, выполняемая на отдельном процессоре и выполняющая действия, предписанные алгоритмом для данного процесса. На втором этапе корневые вершины распределяются по процессам, и каждый из них выполняет обход в глубину

соответствующего поддерева. По окончании обхода своего дерева выполнявший обход процесс освобождается. Для загрузки освободившегося процесса выбирается поддерево, обход которого в этот момент ведется одним из процессов, и в нем выделяется поддерево для обхода освободившимся процессом. При выборе поддерева проверяется некоторое *условие разбиения* дерева, показывающее целесообразность выделения в нем поддерева. Например, максимальный номер полностью пройденного яруса должен быть много меньше числа всех ярусов в дереве. Будем говорить, что дерево, выделенное какому-либо процессу, стало *неделимым*, если для него при обходе перестало выполняться условие разбиения. Если остались лишь неделимые деревья, то освободившийся процесс полностью завершается. Алгоритм заканчивает свою работу при завершении обхода всех поддеревьев.

Выбор процесса и согласование передачи данных между процессами выполняются специально выделенным *управляющим* процессом, функции которого отличаются от функций остальных участвующих в алгоритме *рабочих* процессов. Процессы работают в асинхронном режиме.

Выделение поддерева каким-либо из процессов требует от процесса задержки при обходе. Обход поддерева выполняется методом обхода в глубину. Для выделения поддерева по возможности большего размера ищется первая непройденная вершина наиболее высокого яруса. Такая вершина находится на дне стека, она является корнем непройденного поддерева. Эта вершина удаляется из стека, чем гарантируется, что данное поддерево не будет проходиться дважды, и передается другому процессу. Если у этой вершины есть правый брат, то он заносится на ее место в стек.

Перечисленные операции, а также проверка условия разбиения требуют дополнительного времени, что увеличивает время работы алгоритма.

2.2. Метод назначаемых поддеревьев

Следующий метод параллельного обхода исключает взаимодействие между рабочими процессами. Управляющий процесс выполняет построение и обход дерева в ширину, начиная от его корня, до тех пор, пока число вершин в очереди не достигнет заранее заданного числа *m,* которое *много больше* числа параллельных

процессов *s*. Число *m* определяется в зависимости от размерности задачи и числа процессов. Вершины из очереди посылаются рабочим процессам по мере выполнения теми обходов соответствующих поддеревьев. Таким образом, в то время как один из процессов выполняет обход одного поддерева, другой может обойти несколько поддеревьев. Если очередь пуста, то процесс, завершивший обход очередного дерева, останавливается, в то время как другие могут продолжать работу. В данном алгоритме управление работой процессов не требует таких значительных затрат, как в предыдущем, но существует опасность, что некоторые процессы получат деревья, во много раз превышающие другие по размеру, и в то время, когда они будут продолжать их обход, остальные деревья будут пройдены, и часть процессов будет простаивать. Другими словами, уменьшается загруженность процессоров. Эффективность данного алгоритма зависит от конкретных исходных данных, определяющих вид дерева. Для повышения эффективности можно увеличить число *m*, тогда большие по размеру деревья, возможно, разделятся на две или более частей. Однако слишком большое увеличение числа поддеревьев и, как следствие, уменьшение их размера может привести к частому обращению к управляющему процессу, вследствие чего процессы будут простаивать в очереди, ожидая, пока управляющий процесс обслужит предыдущие запросы.

Взаимодействие процессов в данном алгоритме значительно проще, что важно для многопроцессорных вычислительных систем кластерного типа.

3. Генератор деревьев

Для экспериментального исследования эффективности предложенных методов на большом количестве примеров разработана программа генерации случайных деревьев, порождающая их не целиком, но по мере и в порядке обхода рассматриваемым параллельным алгоритмом. Тип или, точнее, вид дерева характеризуется следующими величинами: глубина дерева – длина максимального пути от корня к листу, ширина дерева – максимальное число узлов, расположенных на одном ярусе, ширина яруса – число узлов расположенных на данном ярусе, степени ветвления вершин.

Эти параметры позволяют генерировать деревья следующих типов (A–D).

A. k-ичное дерево глубины h – дерево, в котором степени ветвления всех внутренних вершин равны константе k. Общее число узлов в таком дереве равно $1+ k+\ldots+ k^r+\ldots+ k^h = O(k^h)$. При $k = 2$ строится бинарное дерево.

Назовем дерево *экспоненциально растущим в ширину*, если число вершин на ярусе растет экспоненциально с ростом номера яруса. k-ичное дерево является таковым.

B. Дерево, степень ветвления любой внутренней вершины которого не меньше k. При $k > 1$ оно является экспоненциально растущим в ширину. Для генерации такого дерева надо задать минимальную степень ветвления k.

C. Обозначим соответственно через w_r и $st(v)$ – максимальное число вершин на ярусе r и степень вершины v. Степени вершин $(r-1)$-го яруса могут выбираться произвольно, но так, чтобы их сумма не превосходила величины w_r. Значения величин w_r могут быть заданы постоянными, выбираться произвольно или задаваться следующими соотношениями, где c_1, c_2, c_3 – натуральные константы:

1) $w_r = c_1 k^r + c_3$ – экспоненциально растущее в ширину дерево;

2) $w_r = c_1 w^{c2}_{r-1} + c_3$ – экспоненциально растущее в ширину дерево;

3) $w_r = c_1 r^{c2} + c_3$ – полиномиально растущее в ширину дерево.

D. Дерево, ширина которого не превосходит константы w.

Предварительное исследование методов с помощью данного генератора показывает, в частности, что на больших k-ичных деревьях их эффективность лежит в диапазоне от 0,8 до 0,98, то есть достаточно близко к единице, и незначительное преимущество имеет метод выделяемых поддеревьев.

Наряду с оценками эффективности параллельных алгоритмов на случайных деревьях интересны также сравнительные оценки их эффективности на деревьях, возникающих при решении конкретных комбинаторных задач – перечислительных, оптимизационных и др. С этой точки зрения определенный интерес представляют такие хорошо известные задачи, как генерация сочетаний и о назначении.

4. Генерация сочетаний

Под генерацией сочетаний здесь понимается порождение всех упорядоченных по возрастанию k-элементных подмножеств заданного n-элементного множества $\{1, 2, \ldots, n\}$, т.е. всех таких векторов $(b_1, b_2,$

…, b_k) с компонентами в {1, 2, …, n}, в которых $b_i < b_{i+1}$, $b_i \le n - k + i$ для $i = 1, \ldots, k$.

Для распараллеливания генерации сочетаний определяется дерево сочетаний – ДС. Его вершинам сопоставляются подмножества множества $N = \{1, 2, \ldots, n\}$, а ребрам – элементы в N. Если некоторой вершине v i-го яруса сопоставлено подмножество Y, то ребрам i-го яруса, исходящим из нее, сопоставляются элементы множества Y, не превосходящие значения $n - k + i$, и, если некоторому ребру, исходящему из вершины v, сопоставлен элемент y, то концу этого ребра – вершине (i+1)-го яруса – сопоставляется множество, полученное исключением из Y всех элементов, не превосходящих y. Корню дерева сопоставляется множество N. Вершины, которым сопоставлены пустые множества, являются концевыми. Глубина ДС равна k. Последовательности элементов, сопоставленных ребрам пути от корня к концевым вершинам и перечисленных в порядке их вхождения в путь, образуют всевозможные сочетания.

Дерево сочетаний интересно тем, что имеет структуру, близкую к структуре деревьев поиска, в том смысле, что степени ветвления вершин одного яруса уменьшаются слева направо, то есть по мере выполнения обхода.

При обходе ДС методом левого обхода в глубину с достижением каждой концевой вершины получается очередное сочетание. Все сочетания перечисляются в лексикографическом порядке.

При параллельном обходе ДС по методу выделяемых поддеревьев поддерево рабочего процесса считается неделимым, если пройдены все его вершины 1-го яруса. Это значит, что с освобождением некоторого процесса выделение поддерева для него происходит у того из процессов, в дереве которого есть непройденная вершина первого яруса; эта вершина и объявляется корнем выделяемого поддерева. Параллельный обход ДС по методу назначаемых поддеревьев осуществляется без особенностей.

5. Задача о назначении

Задача о назначении относится к классу задач поиска оптимального решения, использующих некоторую целевую функцию – характеристику решения. В задачах данного типа в каждой вершине дерева поиска проводится анализ, связанный с полученным на данный момент лучшим решением, по результатам которого

продолжается ветвление дерева из рассматриваемой вершины, либо выполняется возврат на предыдущий уровень.

Задача о назначении состоит в том, что при заданных положительных числах a_{ij} для всех i и j в $N = \{1, 2, \ldots, n\}$ требуется найти перестановку $(k_1, k_2, \ldots, k_n)$ чисел 1, 2,…, n с минимальным значением целевой функции $W = \sum_{i=1}^{n} a_{i,k_i}$ [3].

Дерево поиска решения данной задачи определяется по следующим правилам. Его вершинам сопоставляются подмножества множества N, а ребрам – элементы в N. Если некоторой вершине i-го яруса сопоставлено подмножество Y, то ребрам i-го яруса, исходящим из нее, сопоставляются элементы множества Y, а их концам – вершинам (i+1)-го яруса – множества, полученные исключением из Y элементов, сопоставленных этим ребрам соответственно. Корню дерева сопоставляется множество N. Вершины, которым сопоставлены пустые множества, являются концевыми. Последовательности элементов, сопоставленных ребрам пути от корня к концевым вершинам и перечисленных в порядке их вхождения в путь, образуют всевозможные перестановки множества N.

Для сокращения обхода этого дерева в каждой достигнутой вершине с сопоставленным ей подмножеством $Y = N - \{k_1, k_2, \ldots, k_r\}$ вычисляются число $F(Y) = \sum_{i=1}^{r} a_{i,k_i}$ – *вклад* в значение целевой функции перестановки $(k_1, k_2, \ldots, k_r)$ и нижняя оценка вклада в него перестановки элементов в Y – число $F_0(Y) = \max(A, B)$, где $A = \sum_{i=r+1}^{n} \min_{j \in Y} a_{i,j}$, $B = \sum_{j \in Y} \min_{i=r+1,n} a_{i,j}$. В случае $F(Y) + F_0(Y) \geq W$, где W – достигнутое на данный момент значение целевой функции, эта вершина объявляется бесперспективной. С достижением концевой вершины и соответствующей ей перестановки всех чисел 1, 2, …, n значение целевой функции для последней становится новым значением W [4].

На примере задачи о назначении подчеркнем некоторые особенности параллельных обходов дерева поиска решений

оптимизационных задач, ограничивающие в некоторых случаях возможности для ускорения обхода.

При параллельном обходе дерева, в соответствии с одним из изложенных алгоритмов, в случае получения некоторым из процессов нового лучшего решения, значение целевой функции следует обновить на всех процессорах. Обновление значения целевой функции выполняется следующим образом. Управляющий процесс всегда хранит лучшее на данный момент решение и рассылает соответствующее ему значение целевой функции рабочим процессам. Рабочие процессы периодически проверяют, не передано ли им от управляющего процесса новое решение, и в случае передачи изменяют значение целевой функции на присланное. При построении очередного решения рабочим процессом оно сравнивается с последним присланным значением от управляющего процесса. Если значение, найденное рабочим процессом, лучше, то оно передается управляющему процессу.

В задачах оптимизации ускорение, достигаемое при использовании многопроцессорной системы, зависит от исходных данных, определяющих тип дерева поиска. Например, ускорение может и не превысить 1, независимо от числа процессов. Так, допустим, что решение задачи о назначении есть перестановка, представленная самым левым путем в дереве поиска, то есть перестановка (1, 2, …, *n*). При решении задачи с помощью последовательного алгоритма данная перестановка находится сразу, и не исключено, что, благодаря найденному значению целевой функции, остальные ветви дерева отсекаются уже на первых ярусах, и обход быстро завершается. При использовании параллельных алгоритмов решение будет получено в лучшем случае за то же время, независимо от числа работающих процессов. Возможна и обратная ситуация, в которой решение представлено одним из последних просматриваемых при последовательном алгоритме путей в дереве поиска, но этот путь оказывается первым в поддереве одного из параллельных процессов. Время работы параллельного алгоритма будет складываться из времени, необходимого для построения этого пути, времени, необходимого для передачи найденного значения целевой функции остальным процессам, и времени на вычисление нижних оценок для вершин верхних ярусов. В этом случае ускорение параллельного

алгоритма пропорционально числу узлов просматриваемых в дереве поиска при последовательном обходе.

Литература

1. Кнут Д. Искусство программирования для ЭВМ т.1., М. Мир, 1976. – 734с.
2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.:БХВ-Петербург, 2002. – 608с.
3. Агибалов Г.П., Беляев В.А. Технология решения комбинаторно-логических задач методом сокращённого обхода дерева поиска. – Томск: Изд-во Томского ун-та, 1981. – 125 с.

ОБ ЭФФЕКТИВНОМ РАСПАРАЛЛЕЛИВАНИИ ЧИСЛЕННЫХ МЕТОДОВ ЗАДАЧ РАСПРОСТРАНЕНИЯ ПРИМЕСИ В АТМОСФЕРЕ

Ю.М. Тырчак

chacke@isofts.kiev.ua, chacke@univ.kiev.ua

Современное развитие экологической науки ставит все более сложные вычислительные задачи, решение которых требует применение сложного математического аппарата при построении модели физических процессов и большого количества вычислений. Решение таких задач становится возможным благодаря развитию мультипроцессорных систем и параллельных вычислений [1].

В общем случае модель распространения примесей атмосфере имеет следующий вид:

$$\frac{\partial q}{\partial t} + \sigma q = \frac{\partial}{\partial x_j}\left(\frac{\nu_j}{\xi}\frac{\partial q}{\partial x_j} - v_j q\right),\ (j = 1,2,3),\ \ t_0 \le t \le \tau\,, \tag{1}$$

$$q = q_0(X) \qquad \text{при } t = t_0\,, \tag{2}$$

$$q \to q_\Phi(x_1, x_2, x_3, t) \qquad \text{при } x_1 \to \pm\infty\,,\ x_2 \to \pm\infty\,,\ x_3 \to +\infty\,, \tag{3а}$$

$$q = 0 \qquad \text{при } x_3 = 0 \tag{3б}$$

или

$$\frac{\partial q}{\partial x_3} = 0 \qquad \text{при } x_3 = 0\,. \tag{3в}$$

Однородное граничное условие первого рода (3б) отвечает распространению примесей над сушей, а однородное граничное условие второго рода (3в) – над водной поверхностью; q_Φ – фоновое значение концентрации примесей [4].

Пускай компоненты u, w скорости **V** связаны с координатами x_1, x_3 следующим образом: u – в направлении возростания x_1 (направление разспространения примесей в атмосфере, то есть «вниз»), w – в направлении возростания x_3 (противоположный гравитационной силе, тоисть «вверх»). Тогда модель принимает следующий вид:

$$\begin{aligned} u\frac{\partial u}{\partial x_1} + w\frac{\partial u}{\partial x_3} &= \frac{\partial}{\partial x_3}\left(\nu\frac{\partial u}{\partial x_3}\right), \\ \frac{\partial u}{\partial x_1} + \frac{\partial w}{\partial x_3} &= 0\,, \\ u\frac{\partial q}{\partial x_1} + w\frac{\partial q}{\partial x_3} &= \frac{\partial}{\partial x_3}\left(\nu\frac{\partial q}{\partial x_3}\right). \end{aligned} \tag{4}$$

$$\nu = \chi^2 \frac{\left(\dfrac{du}{dx_3}\right)^3}{\left(\dfrac{d^2u}{dx_3^2}\right)^2}, \tag{5}$$

где $\chi = 0.4$ – эмпирическая константа.

В проекциях на оси (x, z)–координат она выглядит так:

$$\begin{aligned} u\frac{\partial u}{\partial x} + \overline{w}\frac{\partial u}{\partial z} &= \frac{1}{(H-F)^2}\frac{\partial}{\partial z}\left(\nu\frac{\partial u}{\partial z}\right), \\ \frac{\partial u}{\partial x} + \frac{\partial \overline{w}}{\partial z} &= 0\,, \\ u\frac{\partial q}{\partial x} + \overline{w}\frac{\partial q}{\partial z} &= \frac{1}{(H-F)^2}\frac{\partial}{\partial z}\left(\nu\frac{\partial q}{\partial z}\right), \end{aligned} \tag{6}$$

где

$$\overline{w} = \frac{1}{H - F(x)}\left[w - (1 - z)u\frac{\partial F}{\partial x} \right]. \qquad (7)$$

Аппроксимация системы уравнений (6) разностными схемами приводит к системе алгебраических уравнений (в общем случае нелинейных), эффективное решение которых представляет собой сложную проблему. Некоторые трудности возникают при аппроксимации исходных уравнений неявными разностными схемами. Однако именно неявные разностные схемы экономичны при решении уравнений (16), так как они либо не накладывают ограничений на соотношение эволюционного и пространственного шагов сетки, либо значительно ослабляют эти ограничения.

Для численного решения задачи (6) при построении разностных схем можно сочетать различные аппроксимации (симметричные, односторонние) для отдельных членов уравнения. Возьмем на отрезке $0 \le z \le H_{\text{пс}}$ неравномерную с шагами $s_k = z_k - z_{k-1}$, ($k = 1,2,\ldots,K$) сетку. Аппроксимируя пространственные производные первого и второго порядка трехслойными разностными отношениями а производную по x– односторонним разностным отношением «вперед» с постоянным шагом $\ell = x^{n+1} - x^n$, ($n = 0,1,2,\ldots$) и интегрируя численно уравнения (6) на отрезке $t = [t^n, t^{n+1}]$ с помощью формулы трапеции, получим:

$$\frac{u_k^{n+1} - u_k^n}{\ell} = -\frac{1}{2u_k^n}\left\{ \frac{\overline{w}_k^{n+1}}{s_k s_{k+1}(s_k + s_{k+1})}\left[s_k^2\left[u_{k+1}^{n+1} - u_k^{n+1}\right] + s_{k+1}^2\left[u_k^{n+1} - u_{k-1}^{n+1}\right] \right] + \right.$$

$$\left. + \frac{\overline{w}_k^{n}}{s_k s_{k+1}(s_k + s_{k+1})}\left[s_k^2\left[u_{k+1}^{n} - u_k^{n}\right] + s_{k+1}^2\left[u_k^{n} - u_{k-1}^{n}\right] \right] \right\} +$$

$$+ \frac{1}{2u_k^n}\left\{ \frac{s_k\left[\nu_{k+1}^{n+1} + \nu_k^{n+1}\right]\cdot\left[q_{k+1}^{n+1} - q_k^{n+1}\right]}{s_k s_{k+1}(s_k + s_{k+1})} + \right.$$

$$\left. + \frac{s_k\left[\nu_{k+1}^{n} + \nu_k^{n}\right]\cdot\left[u_{k+1}^{n} - u_k^{n}\right]}{s_k s_{k+1}(s_k + s_{k+1})} \right\} - \frac{1}{2u_k^n}\left\{ \frac{s_{k+1}\left[\nu_k^{n+1} + \nu_{k-1}^{n+1}\right]\cdot\left[u_k^{n+1} - u_{k-1}^{n+1}\right]}{s_k s_{k+1}(s_k + s_{k+1})} + \right.$$

$$+\frac{s_{k+1}\left[\nu_k^n+\nu_{k-1}^n\right]\cdot\left[u_k^n-u_{k-1}^n\right]}{s_k s_{k+1}\left(s_k+s_{k+1}\right)}\Bigg\},$$

$$\overline{w_k^{n+1}}=\overline{w_{k-1}^{n+1}}-\frac{s_{k-1}}{2\ell}\left[\left(u_k^{n+1}-u_k^n\right)+\left(u_{k-1}^{n+1}-u_{k-1}^n\right)\right], \qquad (8)$$

$$\frac{q_k^{n+1}-q_k^n}{\ell}=-\frac{1}{2}\left\{\frac{-n+1}{w_k}\frac{s_k^2\left[q_{k+1}^{n+1}-q_k^{n+1}\right]+s_{k+1}^2\left[q_k^{n+1}-q_{k-1}^{n+1}\right]}{s_k s_{k+1}\left(s_k+s_{k+1}\right)u_k^{n+1}}+\right.$$
$$\left.+\frac{-n}{w_k}\frac{s_k^2\left[q_{k+1}^n-q_k^n\right]+s_{k+1}^2\left[q_k^n-q_{k-1}^n\right]}{s_k s_{k+1}\left(s_k+s_{k+1}\right)u_k^n}\right\}+$$
$$+\frac{1}{2}\left\{\frac{s_k\left[\nu_{k+1}^{n+1}+\nu_k^{n+1}\right]\cdot\left[q_{k+1}^{n+1}-q_k^{n+1}\right]}{s_k s_{k+1}\left(s_k+s_{k+1}\right)u_k^{n+1}}+\right.$$
$$\left.+\frac{s_k\left[\nu_{k+1}^n+\nu_k^n\right]\cdot\left[q_{k+1}^n-q_k^n\right]}{s_k s_{k+1}\left(s_k+s_{k+1}\right)u_k^n}\right\}-\frac{1}{2}\left\{\frac{s_{k+1}\left[\nu_k^{n+1}+\nu_{k-1}^{n+1}\right]\cdot\left[q_k^{n+1}-q_{k-1}^{n+1}\right]}{s_k s_{k+1}\left(s_k+s_{k+1}\right)u_k^n}+\right.$$
$$\left.+\frac{s_{k+1}\left[\nu_k^n+\nu_{k-1}^n\right]\cdot\left[q_k^n-q_{k-1}^n\right]}{s_k s_{k+1}\left(s_k+s_{k+1}\right)u_k^n}\right\}$$

Первое и третье уравнения (8) представляют системы алгебраических уравнений с трехдиагональной матрицей, и на каждом эволюционном шаге реализуется с помощью прогонки. Погрешность аппроксимации их при $s = s_k = s_{k+1}$ составляет $\Im = O(\ell^2) + O(s^2)$.

Существующие оценочные приемы, теоретически обоснованные для некоторых достаточно общих модельных задач и проверенные большим опытом использования, позволяют относительно легко доказать абсолютную устойчивость полученных схем.

При численном решении задачи с помощью неявных разностных схем приходим к системе трехточечных разностных уравнений с переменными коэффициентами

$$a_1\vartheta_2+b_1\vartheta_1=f_1, \qquad k=1,$$

$$a_k \vartheta_{k+1} + b_k \vartheta_k + c_k \vartheta_{k-1} = f_k, \quad 2 \le k \le K-1, \qquad (9)$$
$$b_K \vartheta_K + c_K \vartheta_{K-1} = f_K, \quad k = K.$$

Первое и третье уравнения системы (19) получены аппроксимацией граничных условий следующими разностными выражениями:

$$\alpha_1 \frac{\vartheta_2 - \vartheta_1}{s_1} + \beta_1 \vartheta_1 = \chi_1, \; \alpha_2 \frac{\vartheta_K - \vartheta_{K-1}}{s_K} + \beta_2 \vartheta_K = \chi_2,$$

где $s_1 = z_1 - z_0$, $s_k = z_k - z_{k-1}$ – шаги разностной сетки по z, в общем случае – неравномерной.

Во втором уравнении системы (9)

$$a_k = \frac{s_k u_k^{n+1} - \left(\mu_{k+1}^{n+1} + \mu_k^{n+1}\right)}{s_{k+1}\left(s_k + s_{k+1}\right)}, \quad c_k = \frac{s_{k+1} u_k^{n+1} - \left(\mu_k^{n+1} + \mu_{k-1}^{n+1}\right)}{s_{k+1}\left(s_k + s_{k+1}\right)},$$
$$b_k = \frac{1}{h} + \frac{1}{2}\left(a_k + c_k\right), \qquad f_k = \vartheta_k^n \big/ h + \Im_k,$$
$2 \le k \le K-1$.

Здесь $h = x^{n+1} - x^n$ – шаг по эволюционной координате; $\Im_k$ включает аппроксимированные члены уравнения (1–3) на явном слое n – для неявной шеститочечной схемы с погрешностью $O(h^2, s^2)$.

Второе уравнение системы (9) относится ко всем внутренним точкам разностной сетки $1 < k < K$, а граничные условия накладываются в краевых точках $z = 0$ (т.е. $k = 1$) и $z = H$ (т.е. $k = K$).

Матрица системы трехточечных разностных уравнений (9) принадлежит к классу разряженных матриц, в которой из $(K + 1)^2$ элементов ненулевыми являются не более $3K + 1$ элементов. Кроме того, она имеет ленточную структуру (является трехдиагональной матрицей). Такое регулярное расположение ненулевых элементов матрицы позволяет получить очень простые рекуррентные формулы для вычисления решений. Алгоритм, реализующий эти рекуррентные формулы, в литературе по численным методам носит название *метод прогонки,* который представляет собой модифицированный метод исключения Гаусса.

Суть метода прогонки заключается в следующем: *необходимо найти решение задачи* (9) *в рекуррентной форме*

$$\vartheta_{k+1} = E_k \vartheta_k + D_k\,, \quad 1 \le k \le K\,, \tag{10}$$

где E_k и D_k – вспомогательные коэффициенты, значения которых зависят от коэффициентов разностного уравнения и краевых условий (9).

Искомое рекуррентное соотношение, связывающее значения ϑ в точках k и $k-1$ имеет вид: $\vartheta_k = E_{k-1}\vartheta_{k-1} + D_{k-1}$.

Так как соотношения (21) позволяют определить все искомые значения E_k и D_k, то вместе с (10) они задают двойную рекурсивную процедуру решения трехдиагональной системы уравнений по следующему алгоритму.

Граничное условие $\alpha_2 \dfrac{\partial \vartheta}{\partial x} + \beta_2 \vartheta = \chi_2$ при $z = H$ в разностной форме $\vartheta_K = \dfrac{\alpha_2}{\alpha_2 + s_K \beta_2}\vartheta_{K-1} + \dfrac{s_K \chi_2}{\alpha_2 + s_K \beta_2}$, третье уравнение системы (9) $c_K \vartheta_{K-1} + b_K \vartheta_K = f_K$ при $k = K$ и общее рекуррентное соотношение (10) определяют начальные значения E_{K-1} и D_{K-1} в виде

$$E_{K-1} = -\frac{c_K}{b_K} = \frac{\alpha_2}{\alpha_2 + s_K \beta_2}\,, \quad D_{K-1} = \frac{f_K}{b_K} = \frac{s_K \chi_2}{\alpha_2 + s_K \beta_2}\,. \tag{11}$$

Вычислив начальные значения E_{k-1} и D_{k-1}, по формулам (10) последовательно ($k = K-2, K-3, \ldots, 1$) определяются и запоминаются прогоночные коэффициенты E_k и D_k (*прямая прогонка*). Граничное условие $\alpha_1 \dfrac{\partial \vartheta}{\partial x} + \beta_1 \vartheta = \chi_1$ при $z = 0$ в разностной форме $\vartheta_2 = \left(1 - \dfrac{s_1 \beta_1}{\alpha_1}\right)\vartheta_1 + \dfrac{s_1 \chi_1}{\alpha_1}$ и общее рекуррентное соотношение (10) при $k = 1$ определяют первое значение ϑ_1 в виде:

$$\vartheta_1 = \frac{s_1 \chi_1 - \alpha_1 D_1}{s_1 \beta_1 - (1 - E_1)\alpha_1}\,. \tag{12}$$

Вычислив начальное значение ϑ_1, по формуле (10) последовательно ($k = 2, 3, \ldots, K$) ищется решение ϑ_k (*обратная прогонка*). Так как значения ϑ_k находятся здесь последовательно при возрастании индекса k (слева направо), формулы (9)–(12) называют иногда формулами *левой прогонки.*

Сведение численной модели к методу прогонки, предварительно апроксимируя систему уравнений разностными схемами дает довольно существенный прирост в производительности, почти не сказываясь при этом на точности полученых результатов. Распареллеливание задач с использованием метода прогонки уже является достаточно изученой областью программирования [2]. В данное время рассматриваются варианты реализации схемы прогонки на распределенных вычислительных системах с использовапнием программного обеспечения MPI [3].

Литература

1. Дорошенко А.Е. Математические модели и методы организации высокопроизводительных параллельных вычислений. Алгебро-динамический подход. Киев: Наукова думка, 2000. 177 с.
2. Параллельные вычисления / Под ред. Г. Родрига: Пер. с англ. // Под ред. Ю.Г. Дадаева. – М.: Наука, 1986. 376 с.
3. Немнюгин С.А. Стесик О.Л. Параллельноге программирование для многопроцессорных вычислительных систем. СПб: БХВ-Петербург, 2002. 400 с.
4. Довгий С.А., Прусов В.А., Копейка О.В. Использование геоинформационных технологий в системах охраны окружающей среды и исследования природных ресурсов т. 4. К.: Наукова думка, 2000.
5. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.

АРТ – СИСТЕМА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ НА ОСНОВЕ ЯЗЫКА С ДИНАМИЧЕСКИМИ МАССИВАМИ. КОНЦЕПЦИЯ И ЯЗЫК

А.Р. Уразов

Новосибирский государственный технический университет

1. Цели проекта

АРТ – Automatic Parallelization Tool или средство автоматического распараллеливания. Действительно, основной целью проекта является

обеспечение высокой степени автоматизации параллельного программирования численных алгоритмов. Система опирается на характерное свойство численных методов – регулярную обработку больших массивов данных. Именно для таких алгоритмов может быть произведено эффективное автоматическое распараллеливание по данным [1], [2].

Ключевыми при создании системы явились следующие требования:

- сохранение привычной парадигмы последовательного программирования, использование широко распространенного языка;
- неявное с точки зрения программиста использование ресурсов многопроцессорного вычислительного комплекса;
- обеспечение автоматической настройки на ресурсы вычислительной системы, в том числе и выполнение динамической балансировки нагрузки [1].

2. Основные проектные решения

В основу системы программирования положены широко используемые языки программирования C/C++, которые расширяются небольшим набором дополнительных средств, необходимых для генерации эффективных параллельных программ.

Проблема сохранения привычной парадигмы последовательного программирования при использовании многопроцессорных ВС решается введением концепции динамических (или виртуальных) массивов. Понятие виртуального массива – краеугольный камень системы программирования АРТ. Объявляя в программе некоторый массив виртуальным, программист информирует систему о возможности его распределенного хранения и обработки. Синтаксически все операции над виртуальными массивами записываются также как и над обычными, а система сама принимает решение о необходимости распределения. Таким образом, распределенная обработка осуществляется в значительной мере прозрачно для пользователя.

Динамические типы данных, в том числе и динамические массивы, имеют большую историю применения, а их использование ассоциируется с высоким уровнем языка программирования. Так, использование динамических списков лежит в основе языков

искусственного интеллекта LISP [7] и Prolog [8]. Многие языки сценариев, такие как Python [9], позволяют манипулировать динамическими контейнерами, динамические типы данных используются в проблемно-ориентированных языках, таких как SQL или MATLAB. Объединяет все разновидности динамических типов одно свойство – избавление программиста от забот по распределению памяти. В системе АРТ, ориентированной на применение в численных методах, наиболее используемый тип данных – массив. Понятие динамического массива здесь расширяется в соответствии с необходимостью использования параллельных вычислительных систем и скрывает от программиста особенности распределения элементов массива по узлам вычислительной системы. В этом смысле динамические (виртуальные) массивы АРТ сродни аппаратной поддержке виртуальной памяти. В обоих случаях реальное распределение данных маскируется, а доступ осуществляется по логическим адресам.

Наряду с концепцией виртуального массива возникает еще одной понятие – понятие редукционной переменной. Редукционные переменные используются для накопления значений, зависящих от набора элементов виртуальных массивов. Например, для суммирования элементов виртуального вектора нужно использовать редукционную переменную типа «сумма». Аналогично, редукционная переменная типа «сумма» потребуется при вычислении скалярного произведения, если хотя бы один из векторов объявлен как виртуальный.

Для обеспечения гибкой динамической настройки на ресурсы вычислительного комплекса при минимуме затрат во время выполнения программы полномочия по распараллеливанию распределяются между компилятором и системой поддержки времени выполнения.

3. Использование АРТ

С точки зрения пользователя-программиста АРТ представляет собой расширение языков C/C++, содержащее небольшой набор специальных инструкций, которые служат подсказками компилятору для генерации эффективных параллельных программ. Примечательно, что пользователю АРТ не нужно определять взаимодействие между узлами вычислительной системы, как это обычно требуется в системах

параллельного программирования для архитектур с распределенной памятью.

Система АРТ позволяет не только реализовывать программы, в которые параллелизм закладывается изначально, но и поддерживает идею постепенного распараллеливания, когда существующая последовательная программа превращается в параллельную путем незначительных преобразований исходного кода.

Схема использования системы может быть представлена в следующем виде:

1. внесение инструкций параллельной обработки:
   - описание виртуальных массивов;
   - описание редукционных переменных;
2. разбиение программы на секции:
   - инициализация;
   - вычисления;
   - вывод результатов.

Начинается распараллеливание программы с определения виртуальных массивов – массивов, хранение и обработка которых могут быть распределены между всеми узлами параллельного вычислительного комплекса. При определении виртуального массива в программе пользователь указывает по каким размерностям можно производить распределение. Например, для двумерного массива можно указать, что распределение возможно только по первому измерению. В этом случае распределение может производится лишь целыми строками распределяемой матрицы.

Для того чтобы получить существенный выигрыш от распараллеливания обработки виртуальных массивов необходимо, чтобы, во-первых, на практике массив оказывался достаточно большим, и, во-вторых, чтобы его обработка осуществлялась регулярным образом, функционально эквивалентно для множества ячеек массива. Выполнение последнего требования делает возможным распараллеливание по данным.

Следующий шаг после определения виртуальных массивов – описание редукционных переменных. Редукционные переменные помимо типа данных характеризуются еще и типом редукции, который может принимать одно из следующих значений: product, sum, min, max, and, or, xor. Операции, результат которых зависит от цепочек

значений распределяемых массивов, должны определяться с помощью редукционных переменных для повышения эффективности вычислений.

Как правило, большинство вычислений производится при помощи всех узлов вычислительной системы, а ввод-вывод осуществляет отдельный, выделенный узел. Для инструктирования системы о том, что некоторый набор инструкций должен выполняться в режиме ввода (инициализации), вычислений или вывода результатов, для этого блока режим указывается явно.

Чтобы подробно не рассматривать все инструкции языка, использование системы поясняется примером. Для демонстрации на языке C была запрограммирована последовательная версия метода сопряженных градиентов [10] для плотных матриц. Метод широко применяется в современном численном моделировании и вместе с тем достаточно прост, чтобы служить в качестве примера. Исходными данными являются матрица системы A, вектор правой части b и произвольное начальное приближение вектора неизвестных x. Алгоритм вырабатывает приближенное решение x СЛАУ Ax = b. В ходе работы алгоритма используются вспомогательные векторы r, p, ap. Последовательный код приводится в следующем листинге:

```
// матрица СЛАУ
double a[n][n];
// вектор правых частей и вектор неизвестных
double b[n], x[n];
// невязка
double r[n];
// векторы, требуемые алгоритмом
double p[n], ap[n];

// вычисление невязки
for (size_t i = 0; i<n; i++) {
    double t = b[i];
    for (size_t j = 0; j<n; j++) t -= a[i][j]*x[j];
    r[i] = t;
}

double rr = 0;
```

```
for (size_t i = 0; i<n; i++) rr += r[i]*r[i];
while (rr>threshold) {
    // alpha = (r, r)/(Ap, p)
    double app = 0;
    for (size_t i = 0; i<n; i++) {
        double t = 0;
        for (size_t j = 0; j<n; j++) t += a[i][j]*p[j];
        ap[i] = t;
        app += t*p[i];
    }
    double alpha = rr/app;
    // x = x+alpha*p
    for (size_t i = 0; i<n; i++) x[i] += alpha*p[i];
    // r = r-alpha*Ap
    for (size_t i = 0; i<n; i++) r[i] -= alpha*ap[i];
    // beta = (r_new, r_new)/(r_old, r_old)
    double rr_old = rr;
    rr = 0;
    for (size_t i = 0; i<n; i++) rr += r[i]*r[i];
    double beta = rr/rr_old;
    // p = r+beta*p
    for (size_t i = 0; i<n; i++) p[i] = r[i]+beta*p[i];
};
```

Алгоритм представляется последовательностью операций линейного комбинирования векторов, вычислений матрично-векторных произведений и скалярных произведений векторов. Все эти операции отвечают критериям регулярности обработки, и соответствующие массивы могут быть объявлены распределяемыми. При этом операции вычисления скалярного произведения превращаются в операции сборки значений редукционных переменных. Листинг соответствующей параллельной программы приводится ниже:

```
// матрица СЛАУ
dynamic double a[n:distributable][n];
// вектор правых частей и вектор неизвестных
```

```
dynamic double b[n:distributable], x[n:distributable];
// невязка
dynamic double r[n:distributable];
// векторы, требуемые алгоритмом
dynamic double p[n:distributable], ap[n:distributable];

// вычисление невязки
for (size_t i = 0; i<n; i++) {
    double t = b[i];
    for (size_t j = 0; j<n; j++) t -= a[i][j]*x[j];
    r[i] = t;
}

reduction sum double rr;
for (size_t i = 0; i<n; i++) rr = reduce(r[i]*r[i]);

while (rr>threshold) {
    // alpha = (r, r)/(Ap, p)
    reduction sum double app = 0;
    for (size_t i = 0; i<n; i++) {
        double t = 0;
        for (size_t j = 0; j<n; j++) t += a[i][j]*p[j];
        ap[i] = t;
        app = reduce(t*p[i]);
    }
    double alpha = rr/app;
    // x = x+alpha*p
    for (size_t i = 0; i<n; i++) x[i] += alpha*p[i];
    // r = r-alpha*Ap
    for (size_t i = 0; i<n; i++) r[i] -= alpha*ap[i];
    // beta = (r_new, r_new)/(r_old, r_old)
    double rr_old = rr;
    rr = 0;
    for (size_t i = 0; i<n; i++) rr = reduce(r[i]*r[i]);
    double beta = rr/rr_old;
    // p = r+beta*p
    for (size_t i = 0; i<n; i++) p[i] = r[i]+beta*p[i];
```

```
};
```

4. Обзор архитектуры системы

Система состоит из двух компонентов: модуля сборки программ и библиотеки поддержки времени выполнения.

В обязанности модуля сборки входят следующие функции:

1) предобработка программ на АРТ-С/С++:
   - обработка специальных инструкций языка АРТ;
   - анализ и распараллеливание циклов, переменные которых индексируют распределяемые измерения виртуальных массивов;
   - генерация информации о некоторых аспектах размещения виртуальных массивов для более эффективной работы;
2) компилирование полученной программы на С++ и компоновка с библиотекой поддержки времени выполнения.

5. Текущее состояние проекта

На данный момент спроектирована базовая версия языка, определены требования к функциональности библиотеки поддержки и согласован ее интерфейс. Сейчас ведется разработка компилятора и модуля поддержки. Первую комплексную проверку системы планируется провести на задаче моделирования гравитационной динамики многих тел методом частиц в ячейках [2], [3].

6. Резюме

АРТ – высокоуровневая система параллельного программирования, которая сохраняет привычную модель последовательного программирования, значительно упрощая написание параллельных программ. Ключевая идея предложенной системы – использование виртуальных массивов.

Скрывая от программиста особенности архитектур ВС, АРТ позволяет произвести легкий переход от уже готовой последовательной программы к параллельной. Вместе с этим, АРТ накладывает на исходные тексты программ определенные требования, выполнение которых необходимо, чтобы эффективное распараллеливание было возможным. Основное требование системы выражается следующей формулой: «Программист должен выбирать

структуры данных и алгоритмы так, чтобы было возможным распараллеливание по данным».

Литература

1. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. Новосибирск: Наука, 1988.
2. Kraeva M.A., Malyshkin V.E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers. In the International Journal on Future Generation Computer Systems (Elsevier). Vol. 17, issue 6, p.755-765, 2001.
3. Вшивков В.А., Малышкин В.Э., Снытников А.В., Снытников В.Н. Численное моделирование гравитационной динамики многих тел методом частиц в ячейках: параллельная реализация // Сибирский журнал вычислительной математики, № 1, 2003. С. 25-36.
4. Коновалов Н., Крюков В. Параллельные программы для вычислительных кластеров и сетей // Открытые системы, #03/2002. http://www.osp.ru/os/2002/03/012.htm.
5. DVM система. http://dserv.keldysh.ru/dvm/.
6. OpenMP C and C++ Application Program Interface. Version 2.0 March 2002. http://www.openmp.org.
7. David S. Touretzky. Common Lisp: A Gentle Introduction to Symbolic Computation. http://www-2.cs.cmu.edu/~dst/LispBook/index.html.
8. Братко И. Программирование на языке Пролог для искусственного интеллекта / Пер. с англ. – М.: Мир, 1990.
9. Python Language Webpage. http://www.python.org/.
10. Баландин М.Ю., Шурина Э.П. Методы решения СЛАУ большой размерности: Учеб. пособие. – Новосибирск: Изд-во НГТУ, 2000. – 70с.

ДЕКОМПОЗИЦИЯ ПО ДАННЫМ В ЗАДАЧАХ ИТЕРАЦИОННОЙ ОБРАБОТКИ ИЗОБРАЖЕНИЙ НА МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ

В.А. Фурсов, М.А. Дроздов

Самарский государственный аэрокосмический университет им. С.П. Королева
Институт систем обработки изображений РАН, г.Самара

1. Введение

Ряд задач обработки изображений на многопроцессорных системах допускает распараллеливание по данным [1]. В частности, в работе [2] рассматривалась схема разбиения большого изображения на фрагменты при итерационной обработке изображения фильтром с бесконечной импульсной характеристикой (БИХ-фильтром).

В алгоритмах указанного типа для вычисления каждого отсчета на выходном изображении используются отсчеты, как входного, так и выходного изображений в некоторой окрестности. Поэтому процессоры, осуществляющие обработку фрагментов изображения, после каждой итерации должны ожидать данные о значениях отсчетов, непосредственно примыкающих к границам соседних (сопряженных) фрагментов.

С точки зрения удобства организации вычислительного процесса обычно форму фрагментов задают в виде прямоугольников и/или квадратов. Если искажения и соответствующие алгоритмы обработки обладают центральной симметрией, ширина полос в окрестности границ, которыми должны обмениваться процессоры, не зависит от направления границ фрагментов. В этом случае объем передаваемых данных определяется длиной сопряженных границ (для внутренних областей длиной периметра) фрагментов.

При разбиении исходного изображения на квадраты одинаковых размеров для фрагментов, расположенных на границах декомпозируемого изображения, длина границ, сопряженных с соседними фрагментами, а, следовательно, и объем передаваемых данных будет меньше. Указанное различие во времени передачи данных может оказывать существенное влияние на эффективность использования процессоров однородного кластера, если скорость передачи данных низкая. Неэффективность использования

процессоров более заметна, когда число областей, на которые разбивается изображение невелико. Повышение эффективности использования вычислительных ресурсов кластера может быть достигнуто увеличением размеров областей, находящихся на границах и в углах изображения.

Таким образом, возникает задача нахождения такого разбиения исходного изображения, при котором время работы всех процессоров с учетом пересылок сбалансировано.

2. Схема декомпозиции

Рассмотрим случай, когда исходное изображение квадратное $X \times X$, где X – число отсчетов одной стороны изображения. Будем полагать, что для заданных вычислительного алгоритма и вычислительной системы (кластера) известны константы: τ_p – время расчета при обработке одного отсчета изображения и τ_n – среднее время, затрачиваемое на передачу информации, необходимой для одного отсчета изображения. При фиксированных значениях указанных величин наибольшее отношение объема вычислений к объему пересылок достигается при форме фрагмента в виде квадрата.

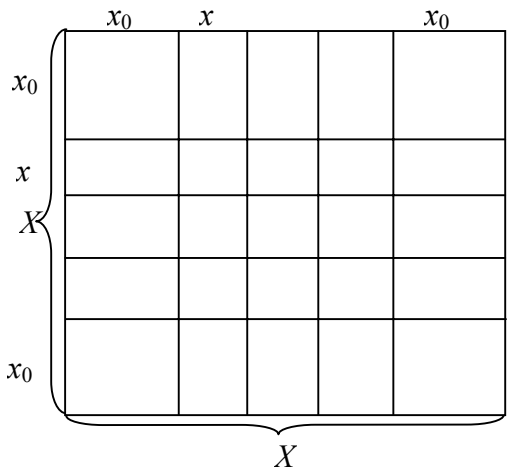
Пусть x – сторона квадратного фрагмента (выраженная числом отсчетов). Потребуем, чтобы величина x удовлетворяла неравенству

$$(4 \cdot \delta)/x \le \Delta_{доп}\ , \qquad (1)$$

где $\delta = \tau_n / \tau_p$ – отношение отрезков времени, необходимых для пересылки данных к времени обработки в расчете на один отсчет изображения, а $\Delta_{доп}$ – допустимая величина отношения времени пересылок к времени обработки внутренней области, задаваемая из условия эффективной загрузки процессоров. Ясно, что неравенство (1) выполняется также для фрагментов, расположенных на границах исходного изображения, т.к. они имеют меньшую длину сопряженных границ. Неравенство только усилится, если размеры этих фрагментов увеличить, не уменьшая размеров внутренних так, как показано на рисунке 1.

Области, находящиеся в углах изображения размером $x_0 \times x_0$, будем называть угловыми, области $x_0 \times x$ ($x \times x_0$) – граничными, а области $x \times x$ – внутренними. Задача заключается в том, чтобы найти x_0 и x такие,

чтобы время обработки всех областей с учетом затрат на пересылку



было одинаковым.

Рис.1. Разбиение квадратного изображения на фрагменты

Если ширина полос на границах областей, которыми они должны обмениваться, мала по сравнению с размерами квадратов, можно в первом приближении считать объем передаваемых данных пропорциональным длине границ (сторон). Тогда суммарное время обработки с учетом пересылок:

а) для внутренней области:

$$T_{вн} = x^2 \cdot \tau_p + 4 \cdot x \cdot \tau_n , \quad (2)$$

б) для граничной:

$$T_{гр} = x \cdot x_0 \cdot \tau_p + 2 \cdot x_0 \cdot \tau_n + x \cdot \tau_n , \quad (3)$$

в) для угловой:

$$T_{угл} = x_0^2 \cdot \tau_p + 2 \cdot x_0 \cdot \tau_n . \quad (4)$$

Положим

$$x_0 = k \cdot x \quad (5)$$

где

$$1 < k < 1{,}5 \quad (6)$$

– коэффициент увеличения угловой (и соответственно граничной) области, который необходимо выбрать из условия балансировки процессоров.

Верхний предел для k указывает значение, при котором балансировка невозможна. В частности, уже при $k = 1{,}5$ длины сопряженных границ, а, следовательно, и объемы передаваемых данных для внутренних и граничных областей одинаковы, а объем вычислений для граничной области больше.

3. Условия балансировки процессоров

Можно показать, что точная балансировка всех процессоров достигается при $\delta = 0$, $k = 1$. Поэтому полную балансировку, при одновременном сокращении времени обработки, можно осуществить лишь для части процессоров. При балансировке процессоров, обрабатывающих внутренние и граничные области, вычислительные затраты на обработку угловых областей существенно возрастают. Поэтому потребуем, чтобы выполнялось равенство

$$T_{угл} = T_{вн}\,. \tag{7}$$

или

$$k^2 \cdot x + 2 \cdot k \cdot \delta = x + 4 \cdot \delta\,. \tag{8}$$

Для выбора k, удовлетворяющего равенствам (7), (8), необходимо исключить величину x. Сделаем это исходя из следующих соображений. Предположим, что удовлетворяющее неравенству (1) $x = (4{\cdot}\delta) / \Delta_{доп}$ является делителем числа X . Тогда при разбиении исходного изображения на одинаковые квадраты ($k = 1$) число полос, на которые может быть разбита одна из сторон X равно X / x. При увеличении сторон угловых квадратных фрагментов в k раз для предельного значения $k = 1{,}5$, число полос n, на которые может быть разбита сторона квадратного изображения длиной X, определяется равенством

$$n = \left(X / x\right) - 1\,. \tag{9}$$

Соотношение для выбора k, полученное из (8) с учетом (9) принимает вид

$$X(k^2 - 1) - (4 - 2k)(n + 1)\delta = 0\,, \tag{10}$$

где X – число отсчетов одной стороны исходного изображения. Заметим, что при k, удовлетворяющем (10), неравенство (1) не нарушается. Дело в том, что число полос n, фигурирующее в (10), определяется равенством (9), которое соответствует предельному значению масштабного коэффициента ($k = 1{,}5$). Поскольку в действительности k в силу (10) обычно оказывается меньшим, величину стороны внутреннего фрагмента необходимо соответственно увеличить. Ясно, что при этом неравенство (1) лишь усиливается. Для заданного в соответствии с (9) n величина $\hat{x}$ стороны квадратного внутреннего фрагмента, уточненная (увеличенная по сравнению с x) с учетом найденного из равенства (10) масштабного коэффициента k определяется как

$$\hat{x} = X/(n + 2\cdot(k-1)). \qquad (11)$$

Остается убедиться, что время обработки граничного фрагмента с учетом пересылок не будет превышать время обработки внутренних и угловых областей:

$$T_{гр} < T_{угл}. \qquad (12)$$

Подставляя в (12) значения $T_{гр}$, $T_{угл}$ из (3), (4) в области допустимых значений k, определяемых неравенствами (6), получаем дополнительное условие

$$k > 0{,}5 + \sqrt{0{,}25 + \delta/\hat{x}}, \qquad (13)$$

где $\hat{x}$ – вычисленная в соответствии с (11) сторона квадратного внутреннего фрагмента.

Для большинства известных итерационных алгоритмов обработки изображений величина $\delta/\hat{x}$ мала, при этом граница (13) почти совпадает с левой границей в (6). Следовательно, практически для всех допустимых k время обработки граничных областей не превышает время обработки внутренних и угловых областей, что приведет к полной загрузке соответствующих процессоров.

4. Обсуждение

Соотношения (9), (10), (11) могут использоваться для выбора начального разбиения обрабатываемого изображения. В действительности эффективность загрузки процессоров будет зависеть также от схемы организации вычислительного процесса и

характеристик коммуникационной среды. Здесь предполагалось, что коммуникационная среда располагает буферной памятью, исключающей простой процессоров из-за неодновременности завершения обработки областей, а сама обработка организована таким образом, что вычисления на очередной итерации могут быть начаты до завершения получения всех данных от сопряженных областей. Выполнение этих условий является отдельной сложной задачей для программиста. Кроме того, в данном случае мы не учитывали латентность при передаче данных, а также тот факт, что X не обязано делиться без остатка на величину $\hat{x}$.

Для преодоления указанных трудностей и более полного учета влияния всех факторов, которые не принимались во внимание в указанной упрощенной постановке, может использоваться технология итерационного планирования распределения ресурсов, описанная в работе [3]. В данном случае ее применение не вызовет дополнительных усложнений по сравнению с вариантом, описанным в указанной работе, поскольку задача выбора k однопараметрическая.

Если исходное изображение не квадратное, то описанный подход может быть применен вначале для разбиения квадратной части изображения со стороной квадрата, равной длине меньшей стороны. Если длина стороны оставшейся части не кратна выбранной величине X, то для декомпозиции остатка также может быть применена технология итерационного планирования распределения ресурсов кластера.

5. Благодарности

Работа выполнена при поддержке Министерства образования РФ, Администрации Самарской области и Американского фонда гражданских исследований и развития (CRDF Project SA-014-02) в рамках российско-американской программы «Фундаментальные исследования и высшее образование» (BRHE,) и РФФИ (гранты № 01-01-00097 №03-01-00109)

Литература

1. Методы компьютерной обработки изображений / Под ред. Сойфера В.А. М.: Физматлит, 2001.

2. Попов С.Б., Сойфер В.А., Тараканов А.А., Фурсов В.А. Кластерная технология формирования и параллельной фильтрации больших изображений // Компьютерная оптика. № 23, 2002. С. 75-78.
3. Фурсов В.А., Шустов В.А., Скуратов С.А. Технология итерационного планирования распределения ресурсов гетерогенного кластера. Труды Всероссийской научной конференции «Высокопроизводительные вычисления и их приложения».- Труды Всероссийской научной конференции г. Черноголовка , 30 октября–2 ноября 2000 г. С.43-46.

ПАРАЛЛЕЛЬНАЯ ВЫЧИСЛИТЕЛЬНАЯ СИСТЕМА С РАСПРЕДЕЛЕНИЕМ ВЫЧИСЛИТЕЛЬНОЙ НАГРУЗКИ МЕЖДУ УЗЛАМИ КОМПЬЮТЕРНОЙ СЕТИ

Н.Н. Хохлов, Р.К. Кудерметов

Запорожский национальный технический университет
khokhlov@zntu.edu.ua, krk@zntu.edu.ua

В последнее время для моделирования сложных задач все чаще применяются параллельные алгоритмы. Как правило, реализуются такие алгоритмы на параллельных суперкомпьютерах, таких как CRAY, HP, SUN и др. [7]. Также все большее распространение получают относительно недорогие параллельные системы на базе персональных компьютеров и основанные на сетевых интерфейсах передачи сообщений (MPI, PVM, MPICH и др.) [1,2], аналогов интерфейсов взаимодействия между процессорами суперкомпьютеров. На сегодняшний день имеется достаточное количество реализаций таких суперкомпьютеров, некоторые из них входят в число самых мощных, которые объединяют вычислительные ресурсы нескольких вычислительных машин, соединенных высокопроизводительными сетевыми интерфейсами, такими как Fast Ethernet, Gigabit Ethernet, SCI. Использование унифицированных средств взаимодействия между узлами вычислительного кластера способно обеспечить объединение в единую систему не только подобных вычислителей, но и компьютеров с различными операционными системами, и с различными архитектурами. Для таких суперкомпьютеров остается актуальной проблема оптимального распределения вычислительной нагрузки

между вычислителями параллельного кластера. Имеющиеся на сегодняшний день спецификации интерфейсов, объединяющих вычислители в единую систему, не имеют явных средств автоматической балансировки вычислительной нагрузки с учетом мощности каждого из вычислителей, что, в свою очередь, приводит к неэффективности распределения параллельной задачи и снижению производительности кластера в целом.

В докладе предлагается вариант организации параллельной вычислительной системы на базе компьютерной сети с попыткой минимизировать число пересылок между вычислителями кластера и учетом мощности каждого из вычислителей при распределении параллельной задачи. В качестве интерфейса передачи сообщений был выбран стандарт MPI в реализации университета Коимбра (Португалия) на базе платформы Microsoft Windows [2], как наиболее полный аналог данного стандарта платформы Unix (так как стандарт в сетевой реализации изначально появился на базе Unix-систем управляющих суперкомпьютерами).

Сетевая параллельная система состоит из пяти основных частей: терминала заданий, с которого осуществляется взаимодействие с сетевой параллельной системой и, в последствии, получение результатов работы; параллельной модели (задача, оформленная в виде DLL, динамически подключаемая к системе моделей); клиент-вычислитель, загружающий в память параллельную модель и осуществляющий обмен данными с сервером параллельной системы; сервер сегмента вычислений, который распределяет параллельные задания между клиентами-вычислителями и осуществляет сбор конечных результатов; контроллер сетевой параллельной вычислительной системы, который осуществляет контроль над всеми серверами (сегментами) параллельной системы и при необходимости распределяет задания между сегментами. Клиент-вычислитель, сервер сегмента вычислений и контроллер сетевой параллельной вычислительной системы загружаются в момент загрузки операционной системы, а параллельные модели – динамически подгружаются по мере надобности.

Таким образом, задача состоит в определении соответствий между долей параллельного алгоритма и мощностью вычислительного узла, которому эта доля будет предназначена.

Проблеме оптимального распределения вычислительной нагрузки в параллельной системе посвящено большое количество научных работ [3, 4, 5], однако большинство из них имеют достаточно сложные алгоритмы, которые по своим вычислительным затратам сравнимы, а то и превосходят вычислительные затраты на решение самой параллельной задачи. Данный факт также нужно учитывать при построении модели оптимального распределения вычислительной нагрузки между узлами кластера.

При решении данной задачи необходимо иметь полную картину топологии данной вычислительной системы с учетом производительности каждого из входящих в кластер компьютеров. Для этого необходимо иметь информацию о производительности процессоров, объема оперативной памяти, объема свободного дискового пространства, удаленности от других узлов, скорости обмена данными между ними и др. Также следует учитывать и текущую загруженность данного узла системы.

Таким образом, введем в систему ряд критериев, описывающих текущую систему с точки зрения имеющихся вычислительных мощностей, при этом критерии можно представить в виде вектора состояния для каждого из узлов $C_i = \{c_j\}$, $i = 1 \ldots N$, $j = 1 \ldots M$, где N – количество узлов данной параллельной системы, а M – количество учитываемых параметров, описывающих каждый узел. Так же для оптимального распределения параллельной задачи между элементами системы необходимо иметь представление о вычислительной сложности данной задачи, в частности о сложности каждой «параллельной ветви» модели. В результате имеем еще один не мало важный вектор, описывающий реализацию текущей задачи $Z = \{z_k\}$, где k – максимальное разветвление данной параллельной задачи. Решением поставленной задачи распределения будет определение, на каком из узлов вычислительной системы, и какая доля параллельной задачи будет выполняться.

В настоящее время ведется работа над средой параллельных вычислений на базе компьютерной сети с учетом приведенных требований. Разрабатываемая среда позволит обеспечить анализ архитектуры, анализ текущей модели по специально сформированному интерфейсу получения данных о ресурсоемкости каждой из параллельных ветвей, расчет оптимального распределения ветвей между элементами кластера, а также контроль активности

каждого из узлов параллельной системы для перераспределения частей задачи вышедшего из строя узла между оставшимися узлами. Сбор и обработку результатов осуществляет сервер сетевого кластера параллельной системы, в дальнейшем результаты передаются терминалу, с которого было получено задание. Также система имеет канал взаимодействия с другими такими же кластерами для объединения их в единую, более крупную параллельную систему. Такая модель параллельной вычислительной системы имеет многоуровневую архитектуру клиент-сервер. В которой взаимодействие между терминалом и сервером вычислительного кластера осуществляется на базе интерфейса сокетов (Sockets) протокола TCP/IP, что позволяет организовать прием заданий на выполнение в параллельной системе через Internet, а также с компьютеров работающих под управлением различными операционными системами. На данном этапе идет поиск и апробация оптимальных путей решения задачи оптимизации распределения параллельной задачи между вычислительными узлами сетевой параллельной системы, а также ведутся работы над оптимизацией процесса обмена служебной информацией между элементами кластера.

Литература

1. MPI: A Massage-Passing Interface Standard Esprit Project P6643 (PPPE), May 5, 1994, 227p.
2. Argonne National Lab MPICH.NT.1.2.2 http://www.pandora.uc.pt:80/w32mpi.
3. Головкин Б.А. Расчет характеристик и планирование параллельных вычислительных процессов. – М.: Радио и связь, 1983.
4. Кременецкий Г.Н., Гуменюк В.А. Применение нейронных сетей для распределения вычислительной нагрузки / Вісник технологічного університету Поділля. Хмельницкий, 2003. № 3. Т.1. С.102–105.
5. Жабин В.И. Динамическое распределение заданий в параллельных вычислительных системах. / Вісник технологічного університету Поділля. Хмельницкий, 2003. № 3. Т.1. С.119-121.
6. Корнеев В.В. Параллельные вычислительные системы. – М.: Нолидж, 1999. 320 с.

7. Наиболее распространенные современные суперкомпьютеры (http://www.parallel.ru/computers/computers.html).

Согласно описанному выше алгоритму при разделении дерева между 4 процессорами получим следующий результат (рис. 2).

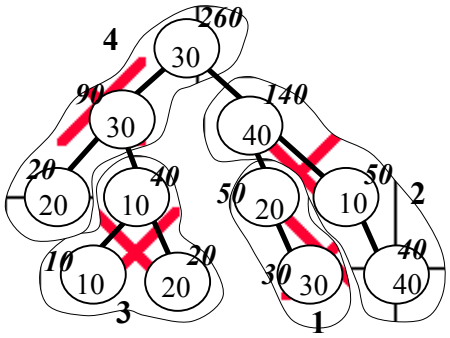


Рис. 2.

Для улучшения балансировки главный корень дерева клик не фиксируется, а рассчитывается на основе специальных формул.

Произведенные расчеты и замеры показывают относительную эффективность предложенной схемы балансировки нагрузки, однако ожидать масштабируемости от параллельной версии для распределенной памяти алгоритма Junction Tree Inference Engine нельзя в силу ограничений, таких как коэффициент распараллеливания, и из-за ожидания активных сообщений. Поэтому дальнейшие исследования связаны с разбиением клик и, как следствие, увеличением обмена информации между процессорами с целью уменьшения взаимного ожидания между ними.

Литература

1. Cowell R., Dawid A., Lauritzen S., Spiegelhaulter D. Probabilistic networks and expert systems, Springer-Verlag, New York, 1999.
2. Pearl J. Probabilistic inference in intelligent systems, Morgan Kaufman, San Mateo, California, 1988.

3. Murphy K., Weiss Y., Jordan M. Loopy belief for approximate inference: an empirical study Proc. of Uncertainty in Artificial Intelligence, 1999.
4. Weiss Y., Freeman W. On the optimality solution of the max-product belief propagation algorithm in arbitrary graphs, IEEE Transactions on Information Theory, 2001, Vol. 47, N2, 723-735.
5. Kozlov A., Singh J. A parallel Lauritzen-Spiegelhalter algorithm for probabilistic inference, IEEE Transactions on Information Theory, 1994, 320-329.

МАСШТАБИРУЕМОСТЬ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ НА ВЕРОЯТНОСТНЫХ СЕТЯХ
(на примере алгоритма Gibbs Sampling)

Р.В. Виноградов, В.П. Гергель, А.В. Сенин

Нижегородский государственный университет им. Н.И.Лобачевского

Введение

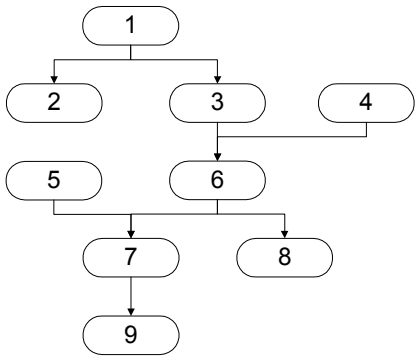
Данная статья посвящена алгоритму Gibbs Sampling, который используется для приближенного вычисления маргинального распределения вероятностей случайных величин на вероятностных сетях. Для знакомства с данной тематикой и приложениями мы рекомендуем книги [1], [2], некоторые понятия и определения рассматриваются в статье [5] этих материалов. Метод Gibbs Sampling дает приближенное распределение вероятностей, которое сходится к точному распределению с ростом числа итераций [4] при достаточно естественных предположениях (в частности требуется, чтобы совместное распределение вероятностей было строго положительным). Необходимость в использовании метода вызвано большой вычислительной трудоемкостью точных алгоритмов, таких как, например, junction tree, в котором с ростом клик и сепараторов мы быстро выходим за пределы возможностей современной вычислительной техники [1]. Тем не менее, несмотря на исключительную эффективность алгоритма, его параллельная реализация как для систем с общей памятью, так и для систем с распределенной памятью вызывает значительный интерес. В данной

статье мы ограничимся параллельной реализацией алгоритма для ориентированных графов.

Последовательный алгоритм

1. Определим термины, использующиеся при описании алгоритма:

Марковское одеяло вершин – множество, включающее сами вершины, всех их родителей, детей и родителей детей. Например, для вершины с номером 6 модели (1) марковское одеяло состоит из вершин с номерами: 3, 4, 5, 6, 7, 8.



(1)

2. *Образец вершин* – набор конкретных значений вершин. Например, для двух бинарных вершин x_1 и x_2, принимающих значения 0 или 1, существует 4 образца: $\{x_1 = 0, x_2 = 0\}$, $\{x_1 = 0, x_2 = 1\}$, $\{x_1 = 1, x_2 = 0\}$, $\{x_1 = 1, x_2 = 1,\}$.

Постановка задачи:

Пусть дана вероятностная сеть A. Требуется найти маргинальное распределение вероятностей на вершинах $\{x_1,\ldots,x_n\}$ модели A: $P\{x_1,\ldots,x_n\}$.

Задаваемые значения:

1. Число итераций – **NIters**.
2. Число генерируемых образцов – **NStreams**.
3. Номер, с которого происходит учет статистики – **NBarrier**.

Алгоритм:

1. Нахождение марковского одеяла вершин $\{x_1,\ldots,x_n\}$. **M** – марковское одеяло.
2. Случайная генерация **NStreams** образцов вершин марковского одеяла **M**. **Streams** – множество образцов.
3. Цикл по всем образцам **Streams**. **s** – текущий образец.
   a. Цикл по числу итераций **NIters**. **i** – номер текущей итерации.
      i. Цикл по номерам вершин **M** образца **s**. **m** – текущая вершина.
         - Получение маргинального распределение вероятностей на вершине **m**, при этом значения вершин – родителей берутся из образца **s**. **P(m, s)** – распределение вероятностей на вершине **m**, с учетом значений вершин - родителей из образца **s**.
         - Генерация нового значения вершины **m** с учетом распределения **P(m,s)**. Запись полученного значения в образец **s**.
      ii. Если (**i** >= **NBarrier**), то добавление образца **s** в множество **Statistics**.
4. Вычисление маргинального распределения вероятностей вершин $\{x_1,\ldots,x_n\}$, используя множество **Statistics**. Вероятность, что вершины $\{x_1,\ldots,x_n\}$ примут некоторые значения $\{a_1,\ldots,a_n\}$ вычисляется следующим образом:

$$P\{x_1 = a_1,...,x_n = a_n\} = \frac{N(\{x_1 = a_1,...,x_n = a\}, Statistics)}{|\,Statistics\,|},$$

где $N(\{x_1 = a_1,\ldots,x_n = a_n\}, Statistics)$ – число элементов множества **Statistics**, в которых $\{x_1 = a_1,\ldots,x_n = a_n\}$, $|Statistics|$ – мощность множества **Statistics**.

Идея распараллеливания алгоритма

Алгоритм Gibbs Sampling допускает распараллеливание на уровне образцов. Так как построение каждой из цепочек образцов[5] происходит совершенно независимо, то это позволяет распределить операции между разными вычислительными узлами. Более того, это

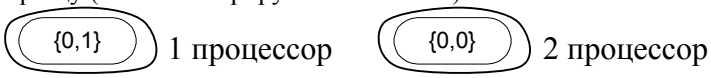
[5] Под цепочкой образцов мы понимаем множество, полученное из некоторого образца, сгенерированного на шаге 2 и всех его «предков», добавляемых в массив **Statistics** на шаге 3.a.ii.

позволяет уменьшить зависимость от выбора начальных данных [3]. Согласование данных требуется только на шаге 4, где для вычисления вероятностей необходимо обладать полной статистикой (знать все сгенерированные образцы). Обычно, шаг 4 занимает время существенно меньшее, чем шаги 1-3, что позволяет получить хорошие коэффициенты распараллеливания.

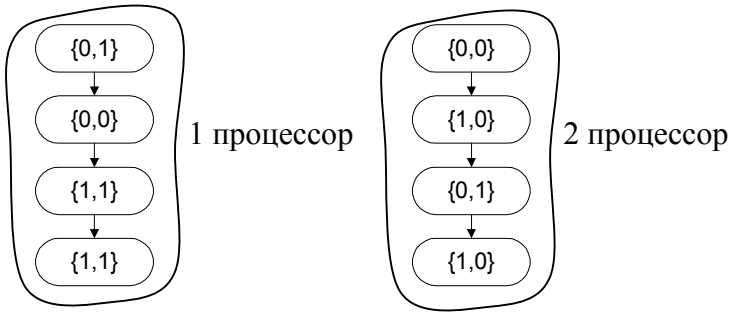
Поясним идею распараллеливания на примере.

Пусть нам требуется получить маргинальное распределение вероятностей на вершинах $\{x_1, x_2\}$ (вершины бинарны). Рассмотрим работу алгоритма.

1. Нахождение марковского одеяла. Положим для простоты, что марковское одеяло вершин $\{x_1, x_2\}$ состоит только из них самих.
2. Случайная генерация двух образцов вершин марковского одеяла (для простоты положим NStreams равным 2). В параллельном варианте алгоритма каждый процессор возьмет себе по одному образцу (он же сгенерирует их значения)



3. Цикл по всем образцам Streams
   a. Генерация новых образцов…
   b. …



   …
4. Вычисление итогового распределения вероятностей.

Результаты вычислительных экспериментов

Реализованы версии алгоритма для многопроцессорных (с использованием технологии OpenMP) и кластерных систем (с использованием технологии MPI).

В таблицах приведены результаты вычислительных экспериментов на 2, 4 и 8 процессорах.

Ускорения при использовании MPI-версии алгоритма.

Условия эксперимента:

1. Процессор – Pentium 4, 1300 МГерц
2. Объем оперативной памяти – 256 МБайт
3. Скорость локальной сети – 100 МБит/сек
4. Операционная система – Windows 2000 Professional.

2 процессора.

Номер теста	Название файла модели	Ускорение
1	test70_4_4_156.bnt	1,9847
2	test80_3_3_149.bnt	1,9701
3	test90_4_4_195.bnt	2,0058
4	test100_3_3_190.bnt	1,9772
5	test120_3_3_230.bnt	1,9723
Среднее значение		1,982

4 процессора.

Номер теста	Название файла модели	Ускорение
1	test70_4_4_156.bnt	3,8277
2	test80_3_3_149.bnt	3,8704
3	test90_4_4_195.bnt	3,9096
4	test100_3_3_190.bnt	3,975
5	test120_3_3_230.bnt	3,9665
Среднее значение		3,9098

8 процессоров.

Номер теста	Название файла модели	Ускорение
1	test70_4_4_156.bnt	7,9883

2	test80_3_3_149.bnt	7,8490
3	test90_4_4_195.bnt	7,9575
4	test100_3_3_190.bnt	7,8224
5	test120_3_3_230.bnt	7,7949
Среднее значение		7,8824

Ускорения при использовании OpenMP-версии алгоритма.

Условия эксперимента:

1. 2 процессора Pentium-3, 1000 МГерц
2. Объем оперативной памяти – 256 МБайт
3. Скорость локальной сети – 1 ГБит/сек
4. Операционная система – Windows 2000 Professional.

Номер теста	Название файла модели	Ускорение
1	test70_4_4_156.bnt	1,9669
2	test80_3_3_149.bnt	1,9605
3	test90_4_4_195.bnt	1,9581
4	test100_3_3_190.bnt	1,9592
5	test120_3_3_230.bnt	1,9706
Среднее значение		1,9631

Условия эксперимента:

1. 4 процессора Pentium 3, 700 МГерц
2. Объем оперативной памяти – 512 МБайт
3. Скорость локальной сети – 1 ГБит/сек
4. Операционная система – Windows 2000 Professional.

Номер теста	Название файла модели	Ускорение
1	test70_4_4_156.bnt	3,8028
2	test80_3_3_149.bnt	3,8215
3	test90_4_4_195.bnt	3,8203
4	test100_3_3_190.bnt	3,7996
5	test120_3_3_230.bnt	3,8309
Среднее значение		3,815016018

Литература

1. Cowell R., Dawid A., Lauritzen S., Spiegelhaulter D. Probabilistic networks and expert systems, Springer-Verlag, New York, 1999.
2. Pearl J. Probabilistic inference in intelligent systems, Morgan Kaufman, San Mateo, California, 1988.
3. Gelman A., Rubin D. A single series from Gibbs sampling provides a false sense of security, Bayeian Statistic, 4, Clarendon Press, Oxford, United Kingdom, pp. 625-631.
4. Cowles M., Carlin B. Markov chain Monte Carlo convergence diagnostic, Journal of American Statistic Association, Vol.91, pp 883-904.
5. Абросимова О., Белов С., Сысоев А. Балансировка вычислительной нагрузки для параллельных алгоритмов на вероятностных сетях. Материалы конференции ННГУ, Нижний Новгород, 2003.

АЛГОРИТМЫ ПЕРЕДАЧИ СООБЩЕНИЙ В БАЙЕСОВЫХ СЕТЯХ И ПРИНЦИПЫ ИХ РАСПАРАЛЛЕЛИВАНИЯ

А.В. Гергель, А.Н. Чернышова

Нижегородский государственный университет им. Н.И. Лобачевского

В данной работе изучаются алгоритмы передачи сообщений в байесовых сетях. Для обстоятельного знакомства с данными сетями, широко используемыми в разных областях, мы рекомендуем классические книги [1, 2], некоторые же определения даются в статье [6], публикуемой в этих материалах. В то время как статья [6] посвящена в большей части алгоритму вывода junction tree, в данной статье мы, прежде всего, рассматриваем алгоритм belief propagation и его современную модификацию loopy belief propagation, позволяющую работать с циклами [2–4]. В этих алгоритмах решение основной задачи вывода (нахождение вероятностей на заданных ненаблюдаемых вершинах при наличии информации на других наблюдаемых вершинах), которая сводится к умножению матриц на вектора, покомпонентному умножению и нормализации векторов,

формулируется в терминах передачи сообщений между вершинами родителями и вершинами детьми. В результате локальных сообщений обновляется вектор belief, характеризующий текущее распределение вероятности на вершине. Известно, что для ориентированных графов, не содержащих циклов, алгоритм belief propagation решает задачу вывода за количество итераций, равное диаметру графа. Для решения задач вывода на графах, содержащих циклы, широко используется модифицированный алгоритм loopy belief propagation, в котором на каждой итерации каждая вершина обменивается информацией со всеми своими соседями, обновляя текущее состояние вектора belief. Условием остановки в этом случае является достижение некоторого стационарного состояния для совокупностей векторов belief. Некоторые теоретические соображения о корректности алгоритма loopy belief propagation, основанные на идее компенсации ошибок, могут быть найдены в [5]. Практическая же ценность этого алгоритма широко известна.

Хорошо известно, что графические модели некоторых известных практических задач содержат тысячи переменных, и, поэтому построение параллельной версии алгоритма loopy belief propagation для систем, как с общей памятью, так и с распределенной памятью, имеет практическую ценность. В данной работе мы рассмотрим задачу распараллеливания для систем с распределенной памятью. Для того чтобы добиться эффективности распараллеливания в этом случае необходимо сбалансировать нагрузку на процессоры и обеспечить минимальную передачу сообщений между ними. Решение данной задачи предусматривает несколько этапов.

Сначала для подсчета трудоемкости вводится понятие веса вершины в зависимости от входных параметров, которыми являются число состояний данной случайной величины, число родителей вершины и ее потомков (с учетом количества их состояний). Данные параметры позволяют определить количество элементарных арифметических действий, выполняемых на данной вершине и, как следствие, ее сравнительную трудоемкость. Далее выделяется остов графа, который затем разбивается на поддеревья примерно равной суммарной трудоемкости вершин, причем вес вершины рассматривается в исходном графе, а не в остове. С целью уменьшения передачи сообщений рассматривается построение остова

максимального веса. После распределения графа по процессорам вычисления ведутся на каждом из подграфов, причем, если соседняя вершина для некоторого узла находится на другом процессоре, то сформированное сообщение посылается средствами MPI. Также реализован алгоритм оптимизации структуры сообщений между процессорами, позволяющий существенно уменьшить общее время передачи. При тестировании параллельной версии алгоритма для распределенной памяти рассматривали графы, содержащие от сотни до четырех тысяч вершин. Соответствующие средние ускорения для двух и четырех процессоров равны 1,853 и 3,516 (тесты проводились на Pentium 3 Xeon с объемом оперативной памяти 512 Мб и скоростью передачи данных 100Мбит/сек).

Произведенные расчеты и замеры показывают достаточную эффективность предложенной схемы балансировки нагрузки, и неплохую масштабируемсть параллельной версии алгоритма при условии, что количество вершин графа много больше числа процессоров. Вместе с тем необходимо отметить, что в случае произвольных графов существенную роль начинает играть обмен сообщениями между процессорами, так как количество этих сообщений может быть значительным. Это лишь в остове исходного графа мы можем гарантировать единственное сообщение между двумя процессорами на каждой итерации. В худшем случае в произвольном графе на каждой итерации количество сообщений между двумя процессорами может увеличиться на число, равное количеству ребер исходного графа, не попавших в остов. С учетом небольшой трудоемкости вычислений на вершинах мы имеем существенную причину для уменьшения ускорения. Тем не менее, для некоторых важных подклассов вероятностных сетей представляется возможным добиться большего ускорения, как, для случая, когда все случайные величина имеют одинаковое количество состояний (например, все вершины бинарные). В этом случае при разбиении остова на поддеревья можно воспользоваться тем фактом, что на каждое ребро при каждой итерации приходится одинаковая нагрузка. Очевидное увеличение ускорения наблюдается на деревьях из-за минимизации количества сообщений. Разумной минимизации сообщений можно также добиться и для графов, имеющих структуру решетки.

Следует отметить, что этот метод распараллеливания может быть применены и к марковским случайным полям.

Литература

1. Cowell R., Dawid A., Lauritzen S., Spiegelhaulter D. Probabilistic networks and expert systems, Springer-Verlag, New York, 1999.

2. Pearl J. Probabilistic inference in intelligent systems, Morgan Kaufman, San Mateo, California, 1988

3. Murphy K., Weiss Y., Jordan M. Loopy belief for approximate inference: an empirical study Proc. of Uncertainty in Artificial Intellegence, 1999.

4. Weiss Y., Freeman W. On the optimality solution of the max-product belief propaation algorithm in arbitrary graphs, IEEE Transactions on Information Theory, 2001, Vol. 47, N2, 723-735.

5. Weiss Y. Correctness of local probability propagation in graphical model with loops, Neural Computations Vol. 12 (2000) pp. 1-41.

6. Абросимова О., Белов С., Сысоев А. Балансировка вычислительной нагрузки для параллельных алгоритмов на вероятностных сетях. Материалы конференции ННГУ, Нижний Новгород, 2003.

СРАВНИТЕЛЬНАЯ ОЦЕНКА ЭФФЕКТИВНОСТИ СИНХРОННЫХ И АСИНХРОННЫХ РЕКУРСИВНЫХ АЛГОРИТМОВ ГЛОБАЛЬНОЙ ОПТИМИЗАЦИИ*

В.А. Гришагин, Д.Е.Квасов, Я.Д. Сергеев

Нижегородский государственный университет им.Н.И. Лобачевского
Калабрийский университет (Италия)

Рассматривается конечномерная задача оптимизации без ограничений

$$f(y) \to \min, \quad y \in D \subseteq R^N, \tag{1}$$

* Данная работа выполнена при поддержке РФФИ (грант № 01-01-00587) и Минобразования РФ (грант № Е02-1.0-58).

$$D = \{y \in R^N : y_j \in [a_j, b_j],\ 1 \le j \le N\}, \qquad (2)$$

в которой целевая функция $f(y)$ является многоэкстремальной и удовлетворяющей условию Липшица. В задаче (1)–(2) требуется найти ее глобально-оптимальное решение.

Многоэкстремальные задачи оптимизации обладают высокой степенью вычислительной сложности, одним из основных факторов которой является размерность (количество варьируемых параметров). Для преодоления высокой трудоемкости многомерных многоэкстремальных задач используются различные подходы их редуцирования к более простым одномерным задачам [1,2]. Один из таких подходов основан на применении *многошаговой схемы редукции размерности* [1,2,4], согласно которой решение исходной задачи сводится к решению семейства рекурсивно связанных одномерных подзадач. Кратко опишем основную алгоритмическую идею многошаговой схемы редукции.

Введем для $1 \le i \le N–1$ вектора $u_i = (y_1,\ldots, y_i)$, $v_i = (y_{i+1},\ldots, y_N)$ и примем, что $v_0 = u_n = y$. Затем, положив по определению $f^N(y) \equiv f(y)$, построим семейство функций

$$f^i(u_i) = \min\{f^{i+1}(u_i, y_{i+1}) : y_{i+1} \in \Pi_{i+1}\},\ 1 \le i \le N-1,$$

определенных на соответствующих проекциях

$$D_i = \{y \in R^i : y_j \in [a_j, b_j],\ 1 \le j \le i\}.$$

Тогда имеет место основное соотношение

$$\min_{y \in D} f(y) = \min_{y_1 \in \Pi_1} \min_{y_2 \in \Pi_2} \ldots \min_{y_N \in \Pi_N} f(y), \qquad (3)$$

где $\Pi_i = [a_i, b_i] \subset R^1$.

Как следует из (3), для решения задачи (1)–(2) достаточно решить одномерную задачу

$$f^1(y_1) \to \min,\ y_1 \in \Pi_1 \subset R^1$$

При этом каждое вычисление функции $f^1(y_1)$ в некоторой фиксированной точке $y_1 \in \Pi_1$ представляет собой согласно (3) решение одномерной задачи

$$f^2(y_1, y_2) \to \min,\ y_2 \in \Pi_2 \subset R^1$$

Эта задача является одномерной задачей минимизации по y_2, т.к. y_1 фиксировано.

В свою очередь, каждое вычисление значения функции $f^2(y_1, y_2)$ при фиксированных y_1, y_2 требует решения одномерной задачи

$$f^3(u_2, y_3) \to \min, \ y_3 \in \Pi_3 \subset R^1$$

и т.д. вплоть до решения задачи

$$f^N(u_{N-1}, y_N) = f(u_{N-1}, y_N) \to \min, \ y_N \in \Pi_N \subset R^1$$

при фиксированном u_{N-1}.

Окончательно решение задачи (1)–(2) сводится к решению семейства "вложенных" одномерных подзадач

$$f^i(u_{i-1}, y_i) \to \min, \ y_i \in \Pi_i \subset R^1, \qquad (4)$$

где фиксированный вектор $u_{i-1} \in \mathrm{D}_{i-1}$.

В настоящей работе рассматривается алгоритмическая схема [5], обеспечивающая параллельную реализацию многошаговой схемы редукции размерности и позволяющая при решении многомерной задачи (1)–(2) использовать до 2^N параллельно работающих процессоров.

В качестве тестового класса для проведения вычислительных экспериментов был выбран класс недифференцируемых многоэкстремальных функций нескольких переменных, предложенный в работе [6]. Для функций данного класса, генерируемых случайно, известны координаты и значения глобального и всех локальных минимумов, причем возможно управление сложностью генерируемых функций посредством задания числа локальных минимумов и области притяжения глобального минимума.

Для решения одномерных подзадач оптимизации (4) использовались алгоритмы, относящиеся к классу параллельных характеристических методов [3], в которых параллелизм обеспечивается параллельным проведением испытаний (вычислений значений целевой функции). Для сравнения были выбраны две схемы распараллеливания. В синхронной схеме новая итерация (параллельное проведение испытаний несколькими процессорами) выполнялась после завершения всех параллельных процессов предшествующей ей итерации. В асинхронной реализации

освободившийся процессор получал координату нового испытания, не ожидая завершения работы других процессоров.

В эксперименте были рассмотрены двух-, трех- и четырехмерные задачи оптимизации (100 функций для каждой размерности). Результаты эксперимента подтвердили теоретические оценки высокой эффективности распараллеливания вычислений по многошаговой схеме, полученные в работе [4] и продемонстрировали существенное ускорение вычислений в асинхронной схеме в сравнении с синхронным параллелизмом.

Литература

1. Стронгин Р.Г. Численные методы в многоэкстремальных задачах. Информационно-статистический подход. М.: Наука, 1978.
2. Strongin R.G., Sergeyev Ya.D. Global optimization with non-convex constraints: Sequential and parallel algorithms, Kluwer Academic Publishers, Dordrecht, Netherlands. 2000.
3. Strongin R.G., Sergeyev Ya.D., Grishagin V.A. Parallel Characteristical Algorithms for Solving Problems of Global Optimization // Journal of Global Optimization,10, 1997. P. 185–206.
4. Sergeyev Ya.D., Grishagin V.A. Parallel asynchronous global search and the nested optimization scheme, Journal of Computational Analysis & Applications, 3(2), 2001. P. 123–145.
5. Гришагин В.А., Филатов А.А. Параллельные рекурсивные алгоритмы многоэкстремальной оптимизации // Материалы второго Международного научно-практического семинара / Под ред. проф. Р.Г. Стронгина. Нижний Новгород: Изд-во Нижегородского госуниверситета, 2002. С. 88–90.
6. Gaviano M., Kvasov D.E., Lera D., Sergeyev Ya.D. Software for Generation of Classes of Test Functions with Known Local and Global Minima for Global Optimization // (Принято к опубликованию в ACM Transactions on Mathematical Software).

СОДЕРЖАНИЕ